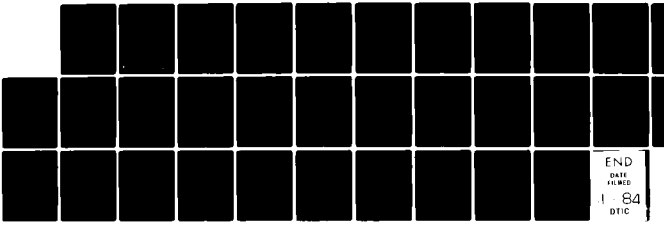
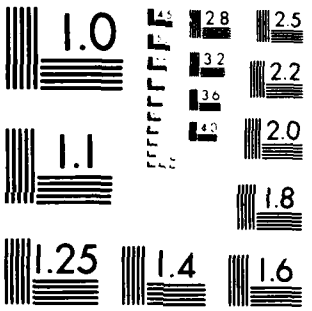
AD-A136 131 USER CENTERED SYSTEM DESIGN: PAPERS FOR THE CHI '1983 1/1
CONFERENCE ON HUMAN..(U) CALIFORNIA UNIV SAN DIEGO LA
JOLLA INST FOR COGNITIVE SCIENCE.. L BANNON ET AL.
UNCLASSIFIED NOV 83 ICS-8305 N00014-79-C-0323 F/G 5/5 NL





ICS REPORT NO. 8305

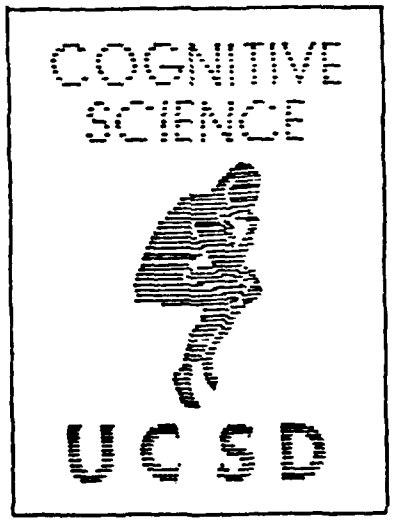
USER CENTERED SYSTEM DESIGN:

PAPERS FOR THE CHI '83 CONFERENCE ON HUMAN FACTORS IN COMPUTER SYSTEMS

Liam Bannon
Eileen Conway
Allen Cypher
Stephen Draper

Janice Graham
Steven Greenspan
Melissa L. Monty
Donald A. Norman

Claire O'Malley
Robert W. Root
Paul Smolensky
Jeffrey Sokolov



INSTITUTE FOR COGNITIVE SCIENCE
UNIVERSITY OF CALIFORNIA, SAN DIEGO LA JOLLA, CALIFORNIA 92093

The research reported here was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and a grant from the System Development Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ONR REPORT 8303

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ONR 8303	2. GOVT ACCESSION NO. AD-A136131	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) User Centered System Design: Papers for the CHI '83 Conference on Human Factors in Computer Systems		5. TYPE OF REPORT & PERIOD COVERED Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) The HMI Project at the University of California, San Diego		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0323
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Human Information Processing Institute for Cognitive Science University of California, San Diego La Jolla, California 92093		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 667-437
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel and Training (Code 442) Office of Naval Research 800 No. Quincy St., Arlington, VA 22217		12. REPORT DATE November, 1983 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was also supported by a grant from the System Development Foundation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

menus	semantic networks	task analysis	multiple perspectives	documentation
command languages	tutorials	user goals	human computer interaction	user needs
response time	structured manual	structured activities	human machine interaction	system modularity
workspace	hypertext	user support	design	software evaluation
questionnaires	mental workload	interleaved tasks	tradeoffs	quick reference
				task-specific help

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

OVER

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

USER CENTERED SYSTEM DESIGN:
PAPERS FOR THE CHI '83 CONFERENCE
HUMAN FACTORS IN COMPUTER SYSTEMS

Abstract

This report includes four papers by the UCSD Project on Human-Computer Interfaces presented at the 1983 Conference on Human Factors in Computer Systems (Boston, December 1983). The first paper, *Evaluation and Analyses of User's Activity Organization* (Bannon, Cypher, Greenspan, and Monty), analyzes the activities performed by users of computer systems. These activities show complex patterns of interleaved activities. This paper develops a framework for discussing the characteristics of these activities in terms of Activity Structures, and provides a number of conceptual guidelines for developing an interface which supports activity coordination.

The second paper, *A Proposal for User Centered System Documentation* (O'Malley, Smolensky, Bannon, Conway, Graham, Sokolov, and Monty), outlines a set of proposals for the development of system documentation based on an analysis of user needs. The paper outlines three specific proposals: a quick-reference facility, a command-line database, and a facility for full explanation and instruction. The paper suggests a way of combining these facilities into an integrated, structured manual, offering more effective user support than is currently provided.

The third paper, *Questionnaires as a Software Evaluation Tool* (Root and Draper), reports on a study investigating the strengths and weaknesses of questionnaires as software evaluation tools. The results suggest that it is important to distinguish between questions addressed to existing features that the users already have experienced and questions about proposed new features, no matter how specific, that they cannot have had experience with. The most successful question type is the checklist which will then give a list of areas needing attention. However, this primarily is useful for evaluating the designer's sins of commission.

The fourth paper, *Design Principles for Human-Computer Interfaces* (Norman), discusses some of the properties that useful principles should have and presents examples of a tradeoff analysis. Any single design technique is apt to have its virtues and deficiencies along different dimensions. Tradeoff analysis provides a quantitative method of assessing tradeoff relations for two attributes by first determining the User Satisfaction function for each, then showing how one trades off against the other. The analysis is used to examine the tradeoff of information versus time, and also of editor workspace versus menu size. Tradeoffs involving command languages versus menu-based systems, choices of names, and handheld computers versus work stations are examined briefly.

ICS REPORT 8305

USER CENTERED SYSTEM DESIGN:

PAPERS FOR THE CHI '83 CONFERENCE ON HUMAN FACTORS IN COMPUTER SYSTEMS

INSTITUTE FOR COGNITIVE SCIENCE
UNIVERSITY OF CALIFORNIA, SAN DIEGO
LA JOLLA, CALIFORNIA 92093

The research reported here was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and a grant from the System Development Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government.

ONR REPORT 8303

EVALUATION AND ANALYSIS OF USER'S ACTIVITY ORGANIZATION

Liam Bannon, Allen Cypher, Steven Greenspan, and Melissa L. Monty
Institute for Cognitive Science
University of California, San Diego

Abstract

Our analyses of the activities performed by users of computer systems show complex patterns of interleaved activities. Current human - computer interfaces provide little support for the kinds of problems users encounter when attempting to accomplish several different tasks in a single session. In this paper we develop a framework for discussing the characteristics of activities, in terms of activity structures, and provide a number of conceptual guidelines for developing an interface which supports activity coordination. The concept of a workspace is introduced as a unifying construct for reducing the mental workload when switching tasks, and for supporting contextually-driven interpretations of the users' activity structures.

This document was jointly authored by the members of the Activity Structures Research Group listed above in alphabetical order. We wish to thank Don Norman, Bob Glushko, and Jonathan Grudin for their insightful comments on earlier drafts of this paper and Tom Erickson for his related ideas and software design.

The Activity Structures Research Group operates under the auspices of the Human-Machine Interface project (UCSD), and is comprised of computer scientists and psychologists. This research is supported by Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Steven L. Greenspan is supported on a Postdoctoral Fellowship by Grant PHS MH 14268 to the Center for Human Information Processing from the National Institute of Mental Health.

Requests for reprints should be sent to HMI, Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA.

To be published in: *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston: December, 1983.

Introduction

The intent of this paper is to give an overview of the work being carried out by the Activity Structures Research Group at UCSD. Our interests are focused upon (a) developing a methodology for studying the complex structuring of activities that often occurs in human-computer interaction, and (b) designing a human-computer interface that is supportive of the user's conception of these multi-level complex activity structures. Our observations of user-computer interactions strongly suggest that the command sequences employed by users are structured and coherent, and that users require computer systems which (a) provide information to reorient the user when resuming tasks which have been interrupted, and (b) minimize the interference incurred when setting up a new task.

We started by analyzing patterns of user-computer interaction. We did this by examining history lists of commands performed by users, aided by a system that reminded users to periodically annotate their lists with descriptions of their intentions during the session. An extract from one of these augmented history lists is provided in Figure 1. Files such as the one shown were collected automatically over a number of sessions with each subject. From the command histories (on the numbered lines) and the user's comments (in < >'s), we were able to discern the users' goals, subgoals and tasks, and how they evolved over time.

The data reveal interesting patterns of commands. Users seem to engage in a number of different activities that can be partitioned into sets of goal-related tasks. For instance, in any single session a user may edit an article, write and debug a program, search for an old file, and answer mail. Often users will work on one task and then, before completing that task, switch to a second one. Tasks tend to be nested within one another; digressions are frequent. When a record is made of these commands as they occur temporally, as in the history list, information on the tasks and goals of the users is lost. An alternative organization of user commands which preserves their task specificity would be useful (see Figure 2). Separating the history list into such functionally distinct units is difficult due to the interleaved activities of the user. In attacking this problem from an Artificial Intelligence perspective, Huff & Lesser (1982) describe a system that utilizes clues in the input patterns to infer the user's activity structures. Our alternative approach assists the user in explicitly indicating the goals associated with each command. Once the user's separate

```
< Write Info Retrieval report.  Its going to take a
long time and be interrupted by other activities>
   15   vi IRreport
< interrupted to prepare a memo.  Send note to gm
about outcome>
   16   snd gm
< back to IRreport>
   17   fg
   18   lf HMI
   19   lf hmi
   20   cdd hmi/IR
   21   ln /csl/norman/IRreport .
   22   ll
< Need to look over the log of old Discussion
Reports.  While at it, decide to trim the log file
of irrelevant stuff that need not be kept, but that
makes file size huge>
   23   msg #hmilog
   24   ll /csl/bin/hmilog
   25   chmod 666 /csl/bin/hmilog
   26   update
< doing Update reminded me of my goals.  Decide to
do the memo >
   27   snd der gm smith
   28   j
   29   lf
   30   cdh
   31   ma
   32   f
< the "f" (finger) showed me funny listing for
smith's address, so ...>
   33   grep smith /etc/passwd
< ok, back to work writing the IR report>
   34   cdpop
   35   lf
   36   vi IR*
< start spelling correction in background>
   37   correct IRreport
   38   bg
< while waiting, read mail and play with numeric
operations in cshell -- triggered by my reading
of the history scripts that S.D. developed>
   39   msg
   40   ps
   41   @ n = 5 + 3
   42   e $n
   43   @n = n * 123
```

Figure 1. Example of an annotated history list showing complex interleaving of activities.

tasks are known, it is possible to treat each task as a separate workspace. The user's pattern of activity may then be characterized as movements from one workspace to another.

The workspace

Our framework for discussing activity structures rests heavily on the notion of a "workspace": an environment dedicated to allowing easy user manipulation of activities to achieve a particular goal or set of functionally related goals. Making the workspace the basic entity in activities coordination has important implications at two levels. From the *users'* perspective, workspaces must have highly dynamic internal structures which can be modified as users reformulate their goals. From the *system's* perspective, a workspace contains tools and data relevant to the users' goals and, in addition, provides a record of the ongoing activities or processes resulting from applying a set of tools to a set of data structures. The internal structure of the workspace thus reflects both the users' goals and the software tools that the computer system can provide for accomplishing these goals.

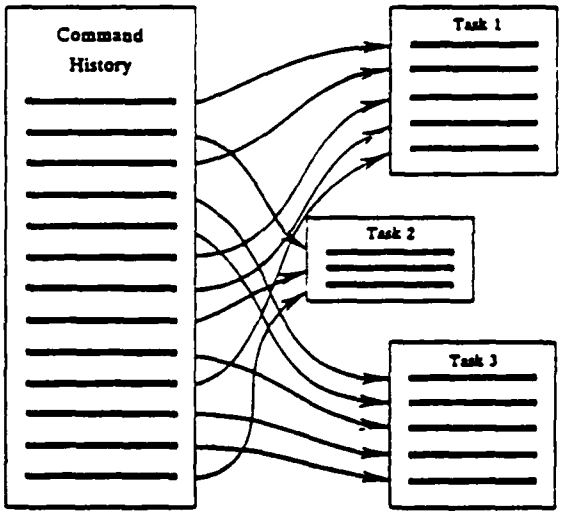


Figure 2. Conceptual model illustrating the reorganization of the command history into funtionally distinct task related command sets.

Workspaces combine the ideas behind functionally-defined *directories* and *workbenches*, [1] but they are more dynamic — they preserve information about the status of activities, and they are capable of being partitioned, recombined, and interrelated by the user. Personal notes and comments on the various files together with a system for displaying information about the workspace organization can be strategically located within the workspace, functioning as memory aids and descriptors of the contents or goals defined by the workspace.

Users encountering a computer system for the first time need models for organizing their activities and accomplishing tasks. They could be provided with a number of "skeletal" workspaces containing a set of tools and examples of their use. Users would be encouraged to build upon the system-provided frameworks by rearranging and personalizing them to suit their needs.

1. A workbench provides an environment which contains a set of programs that are functionally related to a particular type of task (e.g., "writing aids", or "programming").

Activity Coordination Issues

Our investigations into activity structuring resulted in the identification of several classes of activity coordination problems. Based on our analyses of users' annotated command histories, we are suggesting some guidelines for the development of a uniform interface system which should handle the coordination and execution of activities at many levels.

Reducing mental load when switching tasks

A common problem with many interfaces is how to avoid situations where users must struggle with the interface whenever they wish to switch topics. Suppose a user is doing some task but then thinks of an idea relevant to another task. In order for the user to be able to make a note about an idea in the relevant environment, the user must stop the current task, switch directories, open a new file, and, if the idea is still remembered after all this activity, finally type it in. Many users cope with this problem by keeping a pad of paper handy for such occasions. Our proposals are designed to decrease the amount of cognitive overhead — the load on working memory — required for the system interface and to free the user's thinking capacity for the true tasks at hand.

From the user's point of view, entering text should require practically no effort. This implies that naming, organizing, and locating the text within the file system (tasks intended to aid later retrieval) should not impede the activity of entering the text. If necessary, these activities should be delayed indefinitely. The workspace should allow the users to place notes at any position in the current workspace, just as they might "jot down" a note in the margin of a paper. Memory for the context within which the note was created will function as a natural aid to retrieval of the note long after it was written. When the user is ready to devote time to the task, the note could be named, duplicated, and/or relocated. The system would also allow the user to insert reminders and identifiers within the workspace itself, for quick reference on the purpose or context of that workspace.

Suspending and resuming activities

Our observation that users rarely complete any time-consuming activity before beginning another task suggests that a mechanism for easily suspending the current activity is essential. For example, we have found that while doing a task, users often encounter some problem that should be fixed immediately. This means stopping the main task, doing the fix or housekeeping chore, and then resuming the initial task. These contextually-driven activities we call "digressions". While they are common, and often a means of accomplishing many secondary goals which might be forgotten otherwise, they can distract a user from the goals of the main task.

The workspace system should support digressions while providing a large-scale place holder to facilitate easy return to previous activities. Restarting an activity should return the user to the precise state and location within the environment which was frozen, thereby freeing users from the difficult task of remembering what they were doing previously. Some systems currently support this. Berkeley Unix allows tasks to be stopped and resumed, but it does not readily support retention of the full context. Some of the new computer systems with "window" facilities, such as the Xerox STAR (Smith, Irby, Kimball & Verplank, 1982), support certain digressions by allowing users to maintain multiple windows on the screen and to enlarge or shrink these windows and switch between them at will. Saving groups of functionally related windows together as a workspace would help preserve the context of an activity in units which are more meaningful to users.

Maintaining records of activities

The history of a user's activity within a workspace constitutes a record of the command sequence for performing that activity. We call this command sequence an *Activity Script*. Activity scripts are useful whenever a user wants to perform an activity similar to something done before. If the desired activity is exactly the same as a previous one, the user should be able to redo the old activity script. If there are only small differences, the user can edit the script and then execute it. If there are major differences, the old script can still be very useful as a guide to the new activity. No special effort should be involved in creating an activity script, for they are simply records of transactions. Activity scripts could be used to support sophisticated redo/undo facilities which would act on sequences of commands at the task level, rather than just on single command lines.

Functional groupings of activities

We expect that users will want to organize their workspaces to reflect functional groupings. All material associated with a given activity or large group of activities could then be accessed as a unit allowing more continuity between work sessions. For example, the functional relationship between elements of a workspace may be as loosely defined as the group of tasks the user desires to accomplish over the next few sessions. The workspace contents, in this case, function much like a "To Do" list. The potential for hierarchical grouping of workspaces means that this "To Do" workspace might have any number of lower-level workspaces within it that could be maintained in any state of completion, perhaps with notes as reminders of what to do next. When the workspace is reactivated, the user would be shown where work was discontinued, and the activity history list would be available for reference.

Multiple perspectives on the work environment

One consequence of creating complex work environments is that users often lose track of their goals and current location within the system. Multiple window displays do not in themselves solve this problem, as the "messy desk" phenomenon can appear with a vengeance. We address this problem by proposing a set of tools—and ultimately displays—that allow users to organize their activities. The rich interconnections among activities require a support system that allows users to have multiple perspectives on their activities. Such perspectives would include displays indicating activity sequences based on such measures as temporal ordering and goal ordering, this latter being useful in keeping track of the many subtasks necessary to achieve a goal. Possibly other information (eg., the temporal dependencies between tasks) could also be presented. [2]

2. The Apple LISA computer system, for instance, has a PERT program that displays the temporal dependencies existing between different tasks.

Interdependencies among items in different workspaces

Within a complex goal-structured environment, many items (eg. text files and activity scripts) may be important for more than one task. In practice, this means that very often the user may want to assign a portion of one workspace to another workspace. It is therefore important to support the ability to have multiple instances of a particular file or note.

Updating one instance of such an item, however, leads to the question of whether or not to generalize this update to all or several instances. In any viable system the user should be able to trace the relations between workspaces and files and be able to locate multiple instances of a particular item. This facility might enable the system to query the user whenever an item duplicated in other workspaces is updated.

Summary

In our empirical data, the annotations that users added to their history lists showed that they view their interactions with the computer in terms of goals rather than system commands. Accomplishing these goals involves translating them into a series of commands, actually executing the commands, and evaluating the result for success. In the course of this process, users often become confused about their overall goals or lose track of their secondary goals. Our plan is to provide users with a system for managing activities which maps well onto the users' own goal structures, thus reducing the mental load on the users.

Our current research activities focus on both empirical and theoretical developments. We have built a working system (Notepad) to explore in detail the issues involved in how best to postpone certain housekeeping chores while the user is busy creating new text. [3] This system also allows us to explore questions concerning how to manipulate, save and retrieve contexts. In addition, we are developing the concept of a workspace as an integrating idea that has implications for human-computer interface design.

References

Huff, K. E., & Lesser, V. R. Knowledge-Based Command Understanding: An Example for the Software Development Environment. Amherst, Massachusetts: Computer and Information Science, University of Massachusetts, Amherst. June 30, 1982. (Technical Report 82-6.)

Perlman, G. Two Papers in Cognitive Engineering: The design of an interface to a programming system, and MENUNIX: A menu-based interface to UNIX (User manual). La Jolla, California: Center for Human Information Processing, University of California, San Diego. November, 1981. (Report No. 8105.)

Smith, D. C., Irby, C., Kimball, R., & Verplank, B. Designing the Star User Interface. *Byte*, 1982, *7* (No. 4: April), 242 - 282.

Teitelman, W., & Masinter, L., The Interlisp programming environment. *Computer*, *1981*, *14*, (*April*, *No. 4*), 25 - 33.

3. It is useful here to introduce another aspect of the work on activity structures. Cypher has designed a system called Notepad for collecting and organizing textual notes. The principles of organization embodied in Notepad are central in complex activity structuring. A note can be input into the system without first specifying a name or the location of where it should be stored. The note appears as a subset of the current environment and is identified by a number (temporal information). If the note is left uncompleted when the user switches to another task, the system signals the user that there is an uncompleted task and supports reactivating the note for further manipulation. At any time, the note may be named, renamed, or relocated. Notes are hierarchically organized with mechanisms which allow the user to specify the structural family in terms of parents, children, and sibling notes. A note can have multiple parents, meaning that it can be available within any appropriate context and can be easily retrieved.

A PROPOSAL FOR USER CENTERED SYSTEM DOCUMENTATION

C. O'Malley, P. Smolensky, L. Bannon, E. Conway, J. Graham, J. Sokolov, M. L. Monty

Institute for Cognitive Science
University of California at San Diego

Abstract

This paper outlines a set of proposals for the development of system documentation based on an analysis of user needs. It is suggested that existing documentation is not sensitive enough to the variety of levels of user expertise, nor to the variety of contexts in which on-line help is required. We outline three specific proposals for fulfilling these needs: a quick reference facility, a command-line database, and a facility for full explanation and instruction, and suggest a number of ways in which users might access these facilities. Finally, we suggest a way of combining these facilities into an integrated structured manual, offering more effective user support than is currently provided.

Introduction

This paper outlines a project on the development and use of system documentation that is currently underway as a part of the Human Machine Interaction project at UCSD. The goal of the project is twofold: to develop a conceptual framework for discussing the documentation needs of computer users, and to implement a number of software tools that will demonstrably improve the user support currently provided on our UNIX [1] system.

This research was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Requests for reprints should be sent to the Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA.

1. UNIX is a trademark of Bell Laboratories. The comments in this paper refer to the 4.1 BSD version developed at the University of California, Berkeley.

To be published in: *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston: December, 1983.

The need for providing on-line system documentation has become widely accepted (Girill and Luk, 1983; Glushko and Bianchi, 1982). However, with the increasing volume of documentation available on line, users have difficulty finding relevant information at the appropriate level of detail. The on-line documentation facilities outlined in this paper are the first step toward our overall goal of providing a comprehensive, structured system manual that would implement a version of the "hypertext" concept that has been discussed by Nelson (1974). The basic idea is that of structuring information into small, richly interconnected "chunks". Users could be allowed to access the database containing these chunks by a variety of methods, and could browse through the pieces of information or expand on an arbitrary item at will.

The UNIX Environment

UNIX is a very large and powerful operating system comprising over 700 commands. These various programs were built up in a piecewise fashion by a myriad of different programmers, and the available features are a constantly expanding set with little top-down organization imposed on them. The exploitation of the idea of *modular design* of software tools proves to be a very positive feature when viewed from a system perspective. It offers extremely flexible tools, each of which can be used in varied environments, in varied combinations, and for disparate purposes. From the point of view of the user however, this modularity can have disadvantages, especially since it is not directly accessible or explicitly represented. Users have difficulty understanding how the different modules are interconnected and in determining the most effective way to perform a task. Although modularity leads to cutting across program boundaries, standard documentation is still structured around the level of the program. Thus, beginning users with very basic, simple tasks to perform find it difficult to learn how to communicate their requests, and even more difficult to come to a conceptual understanding of the system. Indeed, even experienced programmers are often unaware of the full range of software tools that people have developed for the system, as some of our data explicitly indicate.

The problem of inadequate documentation for the UNIX system has been voiced repeatedly by many people, both at the level of specific program documentation and at the more general level of user support documentation. Existing documentation is not sensitive to the level of expertise of the user. There is a need for a variety of user sup-

port, from the provision of mental models to assist in conceptual understanding to tutorials and on-line help facilities.

One proposal for improving both the quantity and quality of system documentation is to transfer the bulk of this task from programmers and specialist documentation writers to the users themselves. Such an approach has a number of interesting consequences, some of which are explored in Draper (1983). Our philosophy is that the design of documentation is an iterative process of collecting information on user needs, designing an integrated set of tools to support these needs, and evaluating the effectiveness of these tools. We believe that a structured approach to supporting the needs of the user and the design of software tools is necessary to ensure integration of user support tools and appropriate user feedback.

User Needs

We have examined how people use the help facilities existing on our UNIX system by monitoring the use of the on-line manual, and by soliciting feedback from users concerning the information they were trying to obtain. Our analyses have led to the identification of three facilities that users may require:

(1) quick reference;

(2) task specific help;

(3) full explanation.

The need for a quick reference facility was identified from the on-line comments furnished by users as they sought information in the manual. Users said that they had received far more information than they actually wanted, that they were simply trying to confirm the name of a command and its effect, or that they merely wanted to check on the flags or options which the command used. In other words, users need to be able to verify the name of a program, or check on flags, options and syntax, without having to scan through extraneous material.

Task specific or functional needs were identified by requests for help for groupings of commands that perform similar operations or different operations on similar objects. There is a need for help that is sensitive to context or for some kind of database retrieval facility that supports multiple perspectives on the same data.

It is clear that there is often a need for a full description and explanation of the operation of a program. In some cases this might simply require display of the full manual entry on a command. However, other cases might require a relatively lengthy tutorial, for example one explaining the operation of an editor. Based on these observations, we would like to propose a set of potential solutions to these problems.

Quick Reference

Many users who need on-line assistance are already familiar with the commands they wish to use and need only a verification of a command name or a reminder of proper syntax and possible options. Currently they must search through a lengthy manual entry to find this information. Since these users are not interested in a complete explanation of commands, they would be better served by an abbreviated reference manual. To meet this need, we recommend a "Quick Reference" on-line manual that contains only the correct syntax of the command, a list of possible options, and a minimum of explanation. Each quick reference manual entry should also be capable of directing users to other sources of information (i.e., regular manual entry, tutorials, etc.) should they need a more complete explanation.

We have constructed such a quick reference facility for the printing commands available on our system. As an aid to the evaluation of this facility, transcripts of use are being recorded for data analysis. We have been exploring a means of enabling users to provide feedback on the utility of the system that is highly specific yet requires minimal user effort. When users have finished viewing the information on a printing command, they can indicate that that specific use of the quick reference entry was helpful by using an upper case menu selection to continue; the lower case selection achieves the same effect without registering a success. In our preliminary data upper case selections outnumber lower case selections by a significant margin. Furthermore, the data rule out simple case perseveration by individual users, contradicting the expectations of several colleagues. More detailed assessment of the facility must await further data.

Task-Specific Help--A Command Line Database

The concepts that new users bring to bear on a task domain often do not map well onto the concepts incorporated into the system. Thus the user may be able to verbalize the task to be performed but be unable to formulate the corresponding command. A task-specific help facility should address both the need to make explicit the conceptualizations of tasks that are embodied in the system and the need to match the user's descriptions of the task to appropriate commands.

In UNIX, a typical command line may utilize several different programs. It is often difficult for the user to know the various ways programs can interact. We believe this makes the program level inadequate as the sole level for documentation. Since the meaningful units in UNIX are combinations of modules that form command lines, an important level at which to document the system is that of the command line. This would supplement documentation at the program level.

Our monitoring of the use of existing help facilities revealed groupings of commands which reflected a functional organization. Users were seeking help for a particular task by looking for commands which were related to each other in meaningful ways. Having identified the need for a semantic organization in the manual database, we examined the functional units or command lines involved in one particular task domain—that of printing. There are over thirty programs immediately related to printing documents on our system, apart from the preprocessors and postprocessors that are used with these programs. Most of these programs accept at least half a dozen options or flags, depending on the particular task to be performed. Moreover, they can combine with each other in a variety of ways, thus greatly

increasing the number of actual functional units—the command lines. Some meaningful dimensions along which these units are being organized include the type of terminal used, whether hard or soft copy printing is required, or whether formatting, paging or line numbering is required. This large number of dimensions makes the task of organizing a command line database difficult, yet such a facility could be extremely useful both to experienced as well as inexperienced users.

We recommend that each entry in the database contain a command line that performs a specific task (printing or searching, for example), together with an explanation of that task. When several command lines can do a given task, the several synonyms could be included within a single entry, and the subtle differences could be explained. In addition to explanations of the command lines as a whole, *each entry could provide access to existing documentation* on each individual program and its numerous flags. Pointers to relevant existing tutorials could also be included. Finally, a mechanism for browsing among command line entries that perform related tasks could be provided.

The database could be constructed by first analyzing user needs to determine what tasks need to be described, then consulting system experts to determine the appropriate command line components. Several access mechanisms can be provided to present the user with a functionally organized view of the command line database: for example, specially-designed entry through English terms, entries through specification of values for attributes, entry through approximate pattern-matching, or keyword search. We recommend that more than one, if not all, of these access mechanisms be made available.

The mapping between command line entries and users' descriptions should be many-to-many. For example, the word "delete" could access entries for deleting a file, a directory, a command line, a character in a file, and the command line for deleting a file could be accessed by the words "delete", "remove", "rm", "unlink". The access mechanisms should be available not just to provide an initial set of entries in the command line database, but also to pass from one such set to another set containing entries similar to the previous ones but differing in some user-determined respects.

Two mechanisms for generating database entries can be employed. Within task domains that have been characterized *by conceptual dimensions, command lines possessing* the legal combination of attributes can be specifically designed. Secondly, users' command lines can be automatically collected and edited for inclusion in the database, and users can be asked to give a description of the task they hope to perform. The fact that these examples come from users themselves could greatly increase the utility of the database. This data can also be used to compile statistics on the most commonly used commands, enabling default assignments for task dimensions for use when only a few attributes are specified. This would also provide continued evaluation of the facility.

Full Explanation

The command line database does not eliminate the need for a full explanation of the capabilities of an individual program. Existing documentation at the program level should be revised in order to improve the metalanguage for describing syntax, to standardize the structure, to increase consistency of useage of terms, and to improve readability. The command lines in the database that use a given program could be referred to by the documentation on that program, thus greatly increasing the number of examples in the documentation.

Another need exists for the documentation of system features that cross program boundaries. The task dimensions developed for the command line database offer a framework for organizing such documentation. Users could have access to an explanation of any dimensions or attributes chosen when they are performing command line database retrieval. Full explanation of these dimensions could do much to further the user's adoption of appropriate task conceptualizations for the system. This relates to the more general issue of how to introduce new users to the system and how effective these dimensions are in helping the new user form an appropriate mental model of system features.

Integrated Structured Manual

The proposals for projects outlined above draw upon information about UNIX programs that is highly interrelated and intimately bound to the information traditionally contained within a user's manual. As these individual projects reach completion, it should then be possible to draw together the information in the database that each facility uses and develop an integrated database to be accessed by routines that perform the various help functions. This database should contain synopses of programs (as used by the quick reference facility), many examples of complete command lines together with a description of their effects, complete descriptions of programs (as used by a conventional on-line manual), and tutorials on aspects of the system that often cut across program boundaries. The high degree of interconnection between items of information in this database could lend itself to representation as a network; we refer to this network as the "structured manual." This manual will be one realization of Nelson's (1974) vision of "hypertext."

The structured manual would contain a wealth of information about the UNIX system. This information can be broken down into rather small units, such as one-line descriptions of program flags, individual sample command lines, and individual short paragraphs containing tutorial *text, and can be interconnected by network links expressing* mutual relationships. Meta-information can also be contained in the network: information about which tutorial paragraphs are prerequisites of a given paragraph and about which information is appropriate only for novices or of interest only to experts. The network links could be used by a browser that would enable easy passage between related pieces of information, so that good support would be provided for expansion of knowledge about the system.

We believe that these proposals are a positive step towards providing user documentation that is both sensitive to the level of expertise of the user and capitalizes on the flexibility of modular systems.

References

Draper, S. Proposal for a Self-Documenting System. Human Machine Interaction Working Paper, University of California at San Diego (1983).

Girill, T. R., & Luk, C. H. Document: An interactive, online solution to four documentation problems. Communications of the ACM, 26, 5, 328-337 (May 1983).

Glushko, R. J., & Bianchi, M. H. On-line documentation: Mechanizing development, delivery, and use. The Bell System Technical Journal, 61, 6, 1313-1323 (July-August 1982).

Nelson, T. H. *Dream Machines*. South Bend, IN: the distributors, 1974.

Questionnaires as a Software Evaluation Tool

Robert W. Root and Steve Draper
UCSD HMI project
Institute for Cognitive Science C-015
University of California, San Diego
La Jolla, California 92093

Abstract

This paper reports on a study investigating the strengths and weaknesses of questionnaires as software evaluation tools. Two major influences on the usefulness of questionnaire-based evaluation responses are examined: the administration of the questionnaire, and the background and experience of the respondent. Two questionnaires were administered to a large number of students in an introductory programming class. The questionnaires were also given to a group of more experienced users (including course proctors). Respondents were asked to evaluate the text editor used in the class along a number of dimensions; evaluation responses were solicited using a number of different question types. Another group of students received the questionnaire individually, with part of it presented on the computer; a third group also evaluated an enhanced version of the editor in followup sessions.

The order of authors is not significant.

Several people put in a lot of work on this project, most prominently Amy Geoffroy, Jan Graham, and Claire O'Malley. We are also indebted to Professor Kenneth Bowles who offered us a full and generous cooperation in our attempts to invade his Pascal course with our research. A number of other people also contributed to the design and wording of the questionnaires, including: Liam Bannon, Phil Mercurio, Mary Riley, Jeff Sokolov, and Judith Stewart.

This research was conducted under Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research. Work on Human-Computer Interaction was also supported by a grant from the System Development Foundation. Requests for reprints should be sent to Steve Draper, Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA.

To be published in: *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston: December, 1983.

Introduction

A common sentiment expressed by those trying to free themselves of the bad aspects of user-indifferent, programmer-dominated software design is: "Ask the user's opinion of the interface!". The editor questionnaire project described here attempts to explore the usefulness of one obvious interpretation of this maxim – asking for the user's evaluation of a system, in this case a screen-oriented editor, through the use of questionnaires.

Questionnaires and Software Evaluation Methodology. We take the view that software technology should be, and probably is, moving towards greater concern for the user in the design of the interface; the logical endpoint of this process is the adoption of an objective evaluation of the user interface to a program as standard "good practice", in much the same way that well-commented, well-structured code is currently a mark of well-designed software. Realization of this goal will require the development of software evaluation techniques that are 1) inexpensive, 2) easy for software teams to apply, and 3) effective at identifying the good and bad aspects of an interface. While a questionnaire methodology seems promising, as it clearly meets the requirements of inexpensiveness and ease of application, the question of effectiveness remains unanswered. This is the major focus of this study.

A priori there seem to be some grounds for both optimism and pessimism regarding the effectiveness of questionnaire-based evaluations. On the one hand, users are likely to encounter problems that the software designer has not foreseen and any articulation of these problem areas may be adequate to allow the designer to track down and correct the flaws. On the other hand, "naive" users (those with little or no relevant computer experience) may be unable to mount a useful critical response, particularly if such responses must be based either on comparisons with other systems or on the user's own model (built up over time and use) of what interfaces should be like. Experts however can be hard to come by especially for new systems.

or expensive. This study is designed to investigate these uncertainties about the usefulness of "asking the user" through the medium of questionnaires.

Overview of the Study

The major research questions concern:
1) the effects of different types of questions on the quality of the responses; 2) the effects of user experience on the quality of the responses; and 3) the effects of the method of administering the questionnaire on the quality of the responses.

The main test population for the questionnaire study was the introductory programming course in UCSD Pascal at the University of California, San Diego. The course is taken by several hundred students each quarter; for most of them the class is their first significant exposure to computing and interactive editors. The class is self-paced and students are assisted by 20-30 undergraduate proctors who have already completed some instruction in Pascal programming. Thus we had access to a large sample of relatively naive users and, in the same class, a smaller sample of relatively experienced users with whom to compare answers. Furthermore, the UCSD Pascal system is widely used on the campus, affording us access to a larger sample of experienced users. Finally the use of students in an instructional setting as evaluators lends greater generality to the results since this makes it more of a field study than a laboratory one.

We chose to examine the screen editor because editors as a group are probably the most extensively used interactive programs of all, because the editor *must* be used in order to write any program in Pascal, and because editors are heavily interactive and, therefore, contain many of the user-interface qualities we wish to be able to evaluate.

We did not pick the Pascal editor (or the UCSD Pascal system) because it has an exceptional number of flaws, bugs or traps in it. It does not appear to be a bad editor. We picked it for reasons of convenience and familiarity.

We administered a first questionnaire in the first week of the course and a second questionnaire in the seventh week of the course. At about the same time, 20-30 students were given the second questionnaire on an individual basis with part of it implemented on the computer; another group of 20-30 users was exposed to an enhanced version of the Pascal editor to compare their opinions of new editor features before and after exposure. In addition both questionnaires were given to the course proctors at the same time as they were given to the students in the class.

Method

Categories of user experience. Subjects were classified along two independent experience dimensions: experience with the Pascal editor itself, and experience with other editors. The first dimension was provided by the distinction between the students in this course and the proctors (undergraduate teaching assistants). We obtained the latter information from questionnaire one, administered in the first week of class to all students and proctors.

Types of question used. Questionnaire two was designed to elicit information about the user's knowledge of the editor and the weaknesses of its interface. We used three types of questions: *checklists* that list all the editor commands and ask (on 3 point scales) about the respondent's knowledge, use of and problems with each command; *specific questions* about problems with existing features (both commands and other features) and about proposed modifications; and *general (open-ended) questions* about complaints and desired changes to the existing editor (e.g. "are there any (other) commands or traps you would like to see changed in an improved version of the editor?"). The checklists asked on 3-point scales, for each command, whether the user knew it, avoided it, whether it was dangerous, awkward to use, hard to type, had a difficult syntax, or whether it was hard to predict its outcome.

Identifying problem areas. There are two kinds of concerns that a software evaluator may have: to identify problem areas that may not even have been suspected before, and to get more information on existing hypotheses about problem areas. The first concern is addressed by the checklists and the general questions; both yield lists of features perceived as having problems. The respondent's answers may be examined for internal consistency by comparing the lists from each type of question. An important measure of the value of this approach to identifying problem areas is inter-subject agreement (the proportion of subjects identifying each given problem area).

The second concern is addressed by checklists and specific questions about existing features of the editor. Again the answers may be examined for internal consistency by comparing the responses to these specific questions against the checklist responses. The specific questions can also be compared against the general ones for their ability to address features of the editor that cannot be listed on the checklists (which are generated simply from an exhaustive list of commands) -- for instance a non-command feature such as the global direction parameter. (In the Pascal editor this parameter governs the direction of operation of searching and paging commands.) Specific questions also allow the questionnaire user to target particular topics for open-ended discussion by respondents. Again a measure of the usefulness of responses here is agreement between subjects on identifying problems and proposing solutions.

Asking users about proposed solutions. The question types used here are the specific ones, concerning possible changes to the existing editor and asking for rating responses (e.g. "How useful would this change be for you?") and opinions (e.g. "Would you like a delete-word command"), and the general questions dealing with the users' desires for improvements in the editor. The main check on the effectiveness of these questions at eliciting useful responses is provided by the followup study in which 33 respondents were given exposure to an enhanced version of the editor incorporating the proposed changes, and again asked to rate the desirability of the features. A comparison

of pre- and post-exposure responses allows us to judge the reliability of subjects' opinions about proposed changes.

Administering the questionnaire. There is a basic problem in asking people to evaluate an interactive computer system on pencil and paper -- that using the system is something you do, not primarily something you talk about. Thus people may have trouble recalling their editing experiences in response to verbal references to it -- for instance they are probably not practised in thinking of the editor in terms of a set of named commands and features but only in terms of making responses to editing problems. There is a further related problem -- that without a prior set towards thinking of the system with a view to evaluating it, people may not categorize and remember their experiences from this point of view. Thus when you later ask them to do so, they may be unable to recall much of relevance.

We ordered the questions with these considerations in mind, arranging a chance for the user to recall not only the commands (the checklists give an exhaustive list) but also their use. We also had a section of questions testing editor knowledge. These questions required two things of the respondent: producing an editor command sequence that would achieve a given effect (the editor-completion task), and predicting the effect of a given command sequence on a piece of text. The relevance of these questions to the results reported here is only their contributions to reminding the subjects of the experience of using the editor.

To examine the effects on responses of different amounts of support for recall, we used two different conditions for administration of the questionnaire. In the first ("cold") condition the paper and pencil version of the questionnaire was offered to the entire class. Students filled in the questionnaire in their own time, handing it in several days after picking it up. In the second ("hot") condition the editor-completion task, which was done on paper in the "cold" condition, was done on the computer. All other sections of the questionnaire were presented and completed in exactly the same order in both conditions. The hot condition was given to 26 students who signed up for a special session in which they completed the questionnaire and did the computer task. These students also received a token sum ($5.00) for participating in the sessions on their own time.

In the cold condition the editor-completion task was optional. We were led to this by finding in the pilot study that some respondents either failed to fill it out at all, filled it out only partially, or complained vigorously about it. Thus the cold condition was subdivided into "cold" and "ultra-cold", the latter referring to those who chose not to complete the optional section. Out of a class of about 375 students, we eventually recruited 25, 23, and 13 for the hot, cold, and ultra-cold conditions respectively, besides 11 proctors (5 hot, 6 cold).

Results

The checklist questions enabled us first to get a measure of how widely known each command was (see table 1). Besides its intrinsic usefulness this also allowed us to identify which editor commands were used by so few users as to prevent any other significant information to be gained about them. Of the 11 editor commands we judged 3 ("zap", "set", and "find") to be in this category with 69%, 56% and 39% of users stating their ignorance of them respectively.

With the remainder we chose a criterion of problematicity for a command based on the other inquiries in the checklist. This was that (on the 1 to 3 rating scales used throughout the checklists) either the command was avoided or dangerous at least some of the time, or was awkward to use, hard to type, with difficult syntax, or with an outcome difficult to predict all of the time. This criterion was used to decide whether each user flagged the command as problematic, and a rank ordering of commands was made based on the proportion of users flagging it as problematic. These proportions were adjusted for the number of users who claimed to know the command at all (omitting altogether the "zap", "set", and "find" commands which had too few knowledgeable users to be useful). The scores ranged from 89% (replace) to 23% (adjust) (see table 2). The set consisting of the top 3 commands selected by this criteria remained robust across the various experience groups and might be taken as the set in need of attention.

TABLE 1

	subject groups							
command	H	C	U	E	N	S	P	TOTAL
ZAP	83	49	85	55	95	83	18	69
SET	67	38	70	40	83	73	0	56
FIND	50	21	54	35	59	56	0	38
REPLACE	37	21	54	30	35	31	0	26
COPY	10	0	8	5	18	8	0	15
ADJUST	7	3	0	10	6	6	0	8
JUMP	13	3	8	15	6	10	0	8
EXCHANGE	13	0	8	5	12	10	0	7
QUIT	3	0	0	5	0	2	0	1
DELETE	0	0	0	0	0	0	0	0
INSERT	0	0	0	0	0	2	0	0

H=Hot C=Cold U=UltraCold E=Other Editor Experience
N=No Other Editor Experience S=Student P=Proctor

Table 1: Percentage of subjects who don't know each command

TABLE 2

	subject groups							
command	H	C	U	E	N	S	P	TOTAL
REPLACE	84	87	100	86	91	85	91	89
DELETE	80	83	62	90	71	75	91	78
COPY	81	59	67	58	86	86	27	77
QUIT	59	59	38	69	53	47	82	55
EXCHANGE	50	34	25	42	53	42	55	39
INSERT	27	41	23	30	35	32	18	32
JUMP	19	32	25	12	19	28	0	26
ADJUST	11	36	23	17	19	24	9	24

H=Hot C=Cold U=UltraCold E=Other Editor Experience
N=No Other Editor Experience S=Student P=Proctor

Table 2: Percentage of Subjects indicating a problem with each command

One of the specific questions, concerning the direction parameter, could be analyzed in a similar manner despite its different phrasing, in order to provide a way of comparing its effectiveness with the checklist type of question. Users were asked (among other things) how confusing they found it (free response rather than a multi-choice question), and their answers were coded on a 3 point scale. Those coded as "very confusing" were counted as indicating that the feature was problematic. Similarly those who said they still had not learned about it were coded as not knowing the command.

We can divide the subjects along three independent dimensions. The first is by method of administration: "hot", "cold", and "ultra-cold" with 30, 29, and 13 respectively to give a total of 72. The second is by amount of experience of this editor (in our case this coincides with the distinction between proctor and student). The third dimension applies to a subset of 37 for whom we had information on whether they had experience of any other editor – 20 of these did, and 17 did not. We looked at the ordering of questions in terms of how widely they were known and of problematicity for each sub-group separately.

Excluding "zap", "set", and "find", the 3 most problematic commands were "replace", "copy" and "delete" in every group on the "hot" to "ultra-cold" dimension. In the other-editor experience group, "quit" came third with "delete" fourth ("quit" is the fourth most problematic in the population as a whole). If the direction parameter problematicity is compared to other commands it comes last in every group. This does not simply indicate too insensitive a measure since the actual numbers (percent of users finding it problematic) varies from group to group (e.g. 4.7% to 15% for hot and cold) roughly according to the levels of the least problematic command for that group (11% and 32%). Although this study gives no information about how to scale this measure against the other one, it suggests that the two types of question have similar properties.

There were some differences between the groups in the range (discriminative power) of their group judgements of problematicity. The hot group ranged over 73 percentage points, while the cold ranged over 55, and the ultra-cold over 77. The group having more experience of the Pascal editor (the proctors) ranged over 91, while those with less ranged over 62. The group with experience of another editor ranged over 78 points, while those without ranged over 72.

Overall the "general" type of questions produced 118 remarks from 53 individuals out of a total population of 72, and these represented 66 distinct novel comments (i.e. some individuals made essentially the same point as each other). Two measures seem of interest: the percentage of subjects who provided free responses to these questions, and the number of intelligible, relevant and non-redundant responses gleaned (since some responses dealt with matters raised in other questions). The latter measure represents the number of distinct items brought to the designer's attention as a result of the general questions. For the group as a whole, these measures were thus 70% and 66. In comparing subgroups the number of useful responses per member of the group is also relevant. The method of administration produced a substantial difference. The ultra-cold group were the least helpful (only 38% provided any response), presumably reflecting a carry-over from reluctance to do the optional editor task to a reluctance to provide free responses, and yielding only 0.46 useful reponses per person. On the other hand the cold group had a response rate of 86% giving 1.86 useful responses per person, while the hot group with 73% response rate gave 1.36 useful responses per person. Both kinds of experience had a big effect on the number of useful responses: the group with more experience with this editor (proctors) gave 2.18 useful responses per person, while the group with less gave 1.26; and the group with experience of other editors 1.95 while those without gave 0.76 responses per person.

We looked at the responses of a subset of the subjects to questions about 3 features of an enhanced version of the editor before and after they had experience with it, to examine the usefulness of asking respondents about whether they would like some enhancement. The correlation of a group's responses before and after actual experience provided the measure of this. The correlations for all groups were low and with apparently no meaningful variation, the correlation for the whole population being 0.28. Informally it seems that there is more consensus between groups on post-experience than on pre-experience and that within a given group there is a trend to predicting about the same usefulness for all 3 features, while the post-experience scores vary much more.

Our questionnaire afforded two opportunities for internal consistency checks. In one we compared subjects' answers about "zap" on the checklists to their answers to the specific question about its usefulness. 26% percent responded inconsistently that they did not know the command but gave an opinion on its usefulness. (Both hot and cold groups separately showed essentially the same consistency.) In another check we looked at whether, if a subject identified a command as problematic in answer to a general question, they had also done so on the checklist. The group as a whole was 83% consistent by this measure. Only the small proportion of subjects who gave a response to a general question that could be checked in this way.)

Conclusions

Our results suggest that asking users about the value of some proposed change without giving them experience of it is an essentially useless guide to their satisfaction with it in practice. On the other hand our results suggest that checklist-style questions about specific existing features of a system do yield findings that are robust across methods of questionnaire administration and across the amount of user experience, and which are reasonably consistent within subjects. As far as they go, the results support the idea that comparable non-checklist specific questions have the same properties. The robust results from specific questions gave an ordering of commands in terms of how little known they were by users, which is important since an unknown command is of no practical use in an interface. They also

yielded an ordering of trouble spots in the existing system in terms of a measure of user dissatisfaction, together with an estimate of its magnitude that could be used as a guide for a software engineer organizing further work on the system's interface. The non-checklist specific questions addressed to existing editor features showed results similar to the checklists.

The internal consistency checks we were able to do showed that while consistency was high enough not to vitiate the usefulness of the results, it was certainly not 100% by any measure. This indicates the need to build consistency checks into any questionnaire instrument in order to keep track of the quality of the responses. This can be conveniently done by using different question formats referring to the same item, so that while some combinations of response are inconsistent (which yields the consistency check) others give new information about the item.

The different methods of administration, "hot", "cold" and "ultra-cold", did appear to make some difference to the discriminability (range of responses) but did not greatly alter the rank ordering of which commands were least known nor of the troublesomeness of commands. Thus we may tentatively conclude that method of administration may affect the informativeness of results, in the sense that the "colder" the conditions the less likely that any definite information will emerge, but that it does not bias results in the direction of criticizing different commands. It seems therefore that questionnaire administration should perhaps be arranged as much as possible toward "hot" conditions, where the user has as fresh experience of using the system as possible.

The effect of the respondent's experience appears to be similar – it changes the discriminability of the responses, but not the overall results. In our study this was true of experience with the editor in question, and of experience of other editors.

It seems then that the most successful question type is the checklist which will give a list of areas needing attention. However it can only be applied to existing features of the editor – to a designer's sins of commission. A general evaluation tool must also address the problem of gathering information on sins of omission (features that should be added). General (i.e. open-ended questions) might do this. This study did provide evidence that such questions do yield useful responses, which are affected by the factors that affect the specific questions. However a further study, currently being planned, will be needed to discover how *effective such questions are*: for instance if one user complains about a given topic how many users would have complained if asked directly in a specific question?

We began the present study with the view that there might be an important difference in effectiveness between different question types, conceiving of these differences in terms of checklist versus specific versus general. The results suggest that a more important distinction is between questions addressed to existing features that the subjects therefore have experience of, and questions about proposed new features (no matter how specific) that they cannot have had experience of. (This was evident in the results on the predictive power of subjects, and also in the effect of experience on the number of useful responses to general questions.) This view suggests that while general questions might be useful for eliciting criticisms of existing aspects of the editor not mentioned in specific questions, they will only be useful in identifying sins of omission if the subjects have experience of other comparable systems (editors in this case) as a basis for claiming the desirability of adding some feature. This therefore predicts a particular effect of one kind of experience on the usefulness of a particular kind of general question (concerning sins of omission) – again a further study will be needed to address this.

DESIGN PRINCIPLES FOR HUMAN-COMPUTER INTERFACES

Donald A. Norman
Department of Psychology
and
Institute for Cognitive Science C-015
University of California, San Diego
La Jolla, California 92093

ABSTRACT

If the field of Human Factors in Computer Systems is to be a success it must develop design principles that are useful, principles that apply across a wide range of technologies. In the first part of this paper I discuss some the properties that useful principles should have. While I am at it, I warn of the dangers of the tar pits and the sirens of technology. We cannot avoid these dangers entirely, for were we to do so, we would *fail to cope with the real problems* and hazards of the field.

The second part of the paper is intended to illustrate the first part through the example of *tradeoff analysis*. Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another. Tradeoff analysis provides a quantitative method of assessing tradeoff relations for two attributes x_i and x_j by first determining the *User Satisfaction* function for each, $U(x)$, then showing how $U(x_i)$ trades off against $U(x_j)$. In general, the *User Satisfaction* for a system is given by the weighted sum of the *User Satisfaction* values for the attributes. The analysis is used to examine two different tradeoffs of information versus time and editor *workspace versus menu size*. Tradeoffs involving command languages versus menu-based systems, choices of names, and handheld computers versus workstations are examined briefly.

If we intend a science of human-computer interaction, it is essential that we have principles from which to derive the manner of the interaction between person and computer. It is easy to devise experiments to test this idea or that, to compare and contrast alternatives, or to evaluate the quality of the latest technological offering. But we must aspire to more than responsiveness to the current need. The technology upon which the human-computer interface is built changes rapidly relative to the time with which psychological experimentation yields answers. If we do not take care, today's answers apply only to yesterday's concerns.

Our design principles must be of sufficient generality that they will outlast the technological demands of the moment. But there is a second and most important criterion: the principles must yield sufficiently precise answers that they can actually be of use: Statements that proclaim "Consider the user" are valid, but worthless. We need more precise principles.

This new field — Human Factors in Computer Systems — contains an unruly mixture of theoretical issues and practical problems. Just as it is important that our theoretical concerns have breadth, generality, and usability, so too is it important that we understand the practical problems. We are blessed with an exciting, rapidly developing technology that is controlled through the time consuming and addictive procedure called programming. There are traps for the unwary: let me tell you about them.

Tar Pits and Sirens of Technology

As with most unexplored territories, dangers await: tar pits and sirens. The former lie hidden in the path, ready to

To be published in: *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston: December, 1983.

This research was supported by Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Requests for reprints should be sent to Donald A. Norman, Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA.

These ideas have benefited greatly by interactions with the UCSD Human-Machine Interaction project, especially Liam Bannon, Allen Cypher, Steve Draper, Dave Owen, Mary Riley, and Paul Smolensky. Comments on a draft of the paper by Danny Bobrow, Jonathan Grudin, Peter Jackson, Allen Munro, and Julie Norman resulted in major improvements of the ideas and exposition.

trap the unwary. The latter stand openly, luring their prey to destruction with bewitching sweetness. I see too many of you trapped by one or the other.

To program or not to program, that is the question. Whether it is nobler to build systems or to remain pure, arguing for abstract principles independent of the technology. Build systems and you face the tar pits, writing programs whose sole justification is to support the writing of programs, eating up work-years, eating up resources, forever making "one last improvement." When you finish, others may look and nod, saying, "yes, how clever." But will anything general be learned? Will the next technological leap pass it by? Programming can be a pit that grabs the unwary and holds them down. While in the pit they may struggle and attract attention. Afterwards, there may be no visible trace of their passing.

Alternatively, you may be seduced by the sirens of technology. High resolution screens, color, three-dimensions, mice, eye-movement detectors, voice-in, voice-out, touch-in, feelers-out; you name it, it will happen. Superficial pleasure, but not necessarily any lasting result. What general lessons will have been learned?

Damned if you do, damned if you don't. The pure in heart will avoid the struggles, detour the tar pits, blind their eyes to the sirens. "We want general principles that are independent of technology," they proclaim. But then what should they study? If the studies are truly independent of the technology, they are apt to have *little applicability*. How can you develop useful principles unless you understand the powers and weaknesses of the technology, the pressures and constraints of *real design*? Study a general problem such as the choice of editor commands and someone will develop a new philosophy of editing, or a new technological device that makes the old work irrelevant. The problem is that in avoiding the paths that contain the tar, you may never reach any destination; in avoiding temptation, you remain pure, but irrelevant. Life is tar pits and sirens. Real design of real systems is filled with the messy constraints of life: time pressures, budget limitations, a lack of information, abilities, and energy. We are apt not to be useful unless we understand these constraints and provide tools that can succeed despite them, or better, that can help alleviate them. Experimental psychology is not noted for its contributions to life; the study of human-computer interface should be.

Four Strategies for Providing Design Principles

What can we accomplish? One thing that is needed is a way of introducing good design principles into the design stage. How can we do this? Let me mention four ways.

1: Try to impress upon the designer the seriousness of the matter, to develop an awareness that users of systems have special needs that must be taken account of. The problem with this approach is that although such awareness is essential, good intentions do not necessarily lead to good design. Designers need to know what to do and how to do it.

2: Provide methods and guidelines. Quantitative methods are better than qualitative ones, but all are better than none at all. These methods and guidelines must be usable, they must be justifiable, they must have face validity. The designer is apt to be suspicious of many of our intentions. Moreover, unless we have worked out these guidelines with skill, they will be useless when confronted with the realities of design pressures. The rules must not only be justified by reasonable criteria, they must also appear to be reasonable: designers are not apt to care about the discussions in the theoretical journals.

3: Provide software tools for interface design. This can be a major positive force. Consider the problem of enforcing consistent procedures across all components of a system. With appropriate software tools, consistency can be enforced, if only because it will be easier to use the tools rather than to do without them. We can ensure reasonable design by building the principles into the tools.

4: Separate the interface design from other programming tasks. Make the interface a separate data module, communicating with programs and the operating system through a standardized communication channel and language. Interface design should be its own discipline, for it requires sophistication in both programming and human behavior. If we had the proper modularization, then the interface designer could modify the interface independently of the rest of the system. Similarly, many system changes would not require modification of the interface. The ideal method would be for software tools to be developed that can be used in the interface design by non-programmers. I imagine the day when I can self-tailor my own interface, carrying the specification around on a micro-chip embedded in a plastic card. Walk up to any computer terminal in the world, insert my card, and *voila*, it is my personalized terminal.

I recommend that we move toward all of these things. I have ordered the list in terms of my preferences: last being most favored; first being easiest and most likely today. Each is difficult, each requires work.

There has been progress towards the development of appropriate design methods. One approach is demonstrated through the work of Card, Moran, and Newell (1983) who developed formal quantitative methods of assessing a design. Their techniques provide tools for the second of my suggested procedures. Card, Moran, and Newell emphasize the micro-processes of interaction with a computer — for example, analysis at the level of keystrokes. At UCSD, we are attempting to develop other procedures. In the end, the

field will need many methods and guidelines, each complementing and supplementing the others. Let me now describe briefly the approach that we are following, then present one of our techniques — *the tradeoff* analysis — in detail.

The UCSD User Centered System Design Project

At UCSD we have a large and active group attempting to put our philosophy into practice. Our goal is to have pure heart and clear mind, even while feet and loins are in tar and temptation. Some of our initial activities are being presented in this conference: Bannon, Cypher, Greenspan, and Monty (1983); O'Malley, Smolensky, Bannon, Conway, Graham, Sokolov, and Monty (1983); Root and Draper (1983). Other examples have been published elsewhere or are still undergoing final development (Norman, 1983a, b, c).

The principles that we follow take the form of statements plus elaboration, the statements becoming slogans that guide the research. The primary principle is summarized by the slogan that has become the name of the project: *User Centered System Design*. The slogan emphasizes our belief that to develop design principles relevant to building human-machine interfaces, it is necessary to focus on the user of the system. This focus leads us naturally to a set of topics and methods. It means we must observe how people make use of computer systems. It brings to the fore the study of the *mental models* that users form of the systems with which they interact. This, in turn, leads to three related concepts: the designer's view of the system — the *conceptual model*; the image that the system presents to the user — the *system image*; and third, the *mental model* the user develops of the system, mediated to a large extent by the system image. We believe that it is the task of the designer to establish a conceptual image of the system that is appropriate for the task and the class of users, then to construct the system so that the system image guides the user to acquire a mental model that matches the designer's conceptual model.

The Slogans

There are five major slogans that guide the work:

- There are no simple answers, only tradeoffs.
- There are no errors: all operations are iterations towards a goal.
- Low level protocols are critical.
- Activities are structured.
- Information retrieval dominates activity.

There are no simple answers, only tradeoffs. A central theme of our work is that; in design, there are no correct answers, only tradeoffs. Each application of a design principle has its strengths and weaknesses; each principle must be interpreted in a context. One of our goals is to make the tradeoffs explicit. This point will be the topic of the second half of the paper.

All operations are iterations towards a goal. A second theme is that all actions of users should be considered as part of their attempt to accomplish their goals. Thus, even when there is an error, it should be viewed as an attempt by the user to get to the goal. Typing mistakes or illegal statements can be thought of as an approximation. The task for the designer, then, is to consider each input as a starting point and to provide appropriate assistance to allow efficient modification. In this way, we aid the user in rapid convergence to the desired goal. An important implication of this philosophy is that the users' intentions be knowable. In some cases we believe this can be done by having the users state intentions explicitly. Because many commands confound intentions and actions, intentions may substitute for commands.

Low level protocols are critical. By "protocol," we mean the procedures to be followed during the conduct of a particular action or session, this meaning being derived from the traditional meaning of protocol as "a code of diplomatic or military etiquette or precedence." Low level protocols refer to the actual operations performed by the user — button pushes, keypresses, or mouse operation — and these permeate the entire use of the system. If these protocols can be made consistent, then a major standardization takes place across all systems.

Activities are structured. User actions have an implicit grouping corresponding to user goals; these goals may be interrelated in various ways. Thus, a subgoal of a task is related to the main task in a different way than is a diversion, although both may require temporary cessation of the main task, the starting up of new tasks, and eventual return to the main one. We believe the grouping of user goals should be made explicit, both to the user and to the system, and that doing so will provide many opportunities for improved management of the interaction. For example, the system could constrain interpretation of user inputs by the context defined by the current activity, the system could remind users of where they are as they progress through a collection of tasks, or, upon request, it could provide suggestions of how to accomplish the current task by suggesting possible sequences of actions. The philosophy is to structure activities and actions so that the users perceive themselves as selecting among a set of related, structured operations, with the set understood and supported intelligently by the system (see Bannon, Cypher, Greenspan, & Monty, 1983).

Information retrieval dominates activity. Using a computer system involves stages of activities that include forming an intention, choosing an action, specifying that action to the system, and evaluating the outcome. These activities depend heavily upon the strengths and weaknesses of human short- and long-term memory. This means that we place emphasis upon appropriate design of file and directory structures, command "workbenches," and the ability to get information, instruction, and help on the different aspects of the system. We are studying various representational structures, including semantic networks, schema structures of both conventional and "additive memory"

form, browsers, hyper-text structures, and other retrieval aids (see O'Malley, Smolensky, Bannon, Conway, Graham, Sokolov & Monty, 1983).

A Demonstration System

This is where we traverse the tar pits. We feel it essential that our ideas be tested within a working system, not only because we feel that the real constraints of developing a full, useable system are important design considerations that must be faced, but also because we believe that full evaluation can only take place within the bounds of a complete, working environment. Therefore, we intend to construct a test and demonstration system based around a modern workstation using the UNIX operating system. UNIX was chosen because it provides a rich, powerful operating environment. However, because UNIX was designed for the professional programmer, unsophisticated users have great trouble with it, providing a rich set of opportunities for our research.

Although we intend that our design principles will be applicable to any system regardless of the particular hardware being used, many of the concepts are effective only on high-resolution displays that allow multiple windows on the screen and that use simple pointing devices. These displays allow for a considerable improvement in the design of human-computer interfaces. We see no choice but to brave the sirens of technology. The capabilities of the hardware factor into the tradeoff relationships. We intend the demonstration systems to show how the tradeoffs in design choices interact with the technology.

Tradeoffs in Design

Now let us examine one of our proposals — tradeoffs — as a prototype of a quantitative design rule. It is well known that different tasks and classes of users have different needs and requirements. No single interface method can satisfy all. Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another. Each technique provides a set of tradeoffs. The design choices depend upon the technology being used, the class of users, the goals of the design, and which aspects of interface should gain, which should lose. This focus on the tradeoffs emphasizes that the design problem must be looked at as a whole, not in isolated pieces, for the optimal choice for one part of the problem will probably not be optimal for another. According to this view, there are no correct answers, only tradeoffs among alternatives.

The Prototypical Tradeoff: Information Versus Time

One basic tradeoff pervades many design issues:

Factors that increase informativeness tend to decrease the amount of available workspace and system responsiveness.

On the one hand, the more informative and complete the display, the more useful when the user has doubts or lacks understanding. On the other hand, the more complete the display, the longer it takes to be displayed and the more space it must occupy physically. This tradeoff of amount of information versus space and time appears in many guises and is one of the major interface issues that must be handled. To appreciate its importance, one has only to examine a few recent commercial offerings, highly touted for their innovative (and impressive) human factors design that were intended to make the system easy and pleasurable to use, but which so degraded system response time that serious user complaints resulted.

It is often stated that current computer systems do not provide beginning users with sufficient information. However, the long, informative displays or sequence of questions, options, or menus that may make a system usable by the beginner are disruptive to the expert who knows exactly what action is to be specified and wishes to minimize the time and mental effort required to do the specification. We pit the expert's requirement for ease of specification against the beginner's requirement for knowledge.

I approach this problem by tackling the following questions:

- *How can we specify the gain in user satisfaction* that results from increasing the size of a menu;
- How do we specify the user satisfaction for the size of the workspace in a text editor;
- How can we specify the loss in user satisfaction from the increase in time to generate the display and decrease in available workspace;
- How can we select menu size, workspace, and response time, when each variable affects the others?

I propose that we answer the question by use of a psychological measure of *User Satisfaction*. This allows us to determine the impact of changing *physical* parameters upon the *psychological* variable of user satisfaction. Once we know how each dimension of choice affects user satisfaction, then we can directly assess the tradeoffs among the dimensions.

Example: Menu Size and Display Time

Let $U(x)$, the user satisfaction for attribute x, be given by a power function, $U(x) = kx^a$. (In Norman, 1983c, I give more details of the method. See Stevens, 1974, for a review of the power function in Psychology.) For the examples in this paper I used the method of magnitude production to estimate parameters.

User preference for menu size. The preferred amount of information must vary with the task, but informal experiments with a variety of menus and tasks suggest that for many situations, about 300 characters is reasonable: I assigned it a satisfaction value of 50. This is the size menu that can be requested for our laboratory's computer mail program ("msg"). It serves as a reminder for 26 single-letter mnemonic commands. To do the power function estimates,

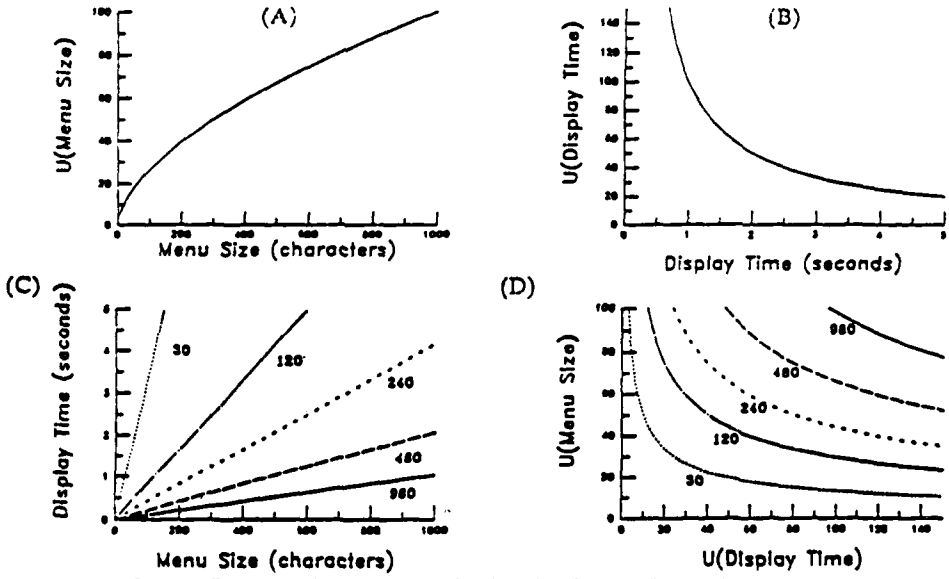


Figure 1. Tradeoff of menu size for display time. Panels A and B show User Satisfaction for menu size, $U(S) = 1.9S^{0.6}$, and display time, $U(T) = 100T^{-1}$, respectively. Panel C shows display time as a function of menu size, $T = S/\beta$, for different values of display rate, β (specified in *characters/second*). Panel D shows the tradeoff between $U(S)$ and $U(T)$ for different values of display rate (β).

I examined a variety of menus of different sizes for the message system (thereby keeping the task the same). I estimated that the menu size would have to increase to half the normal video terminal screen (1000 characters) in order to double my satisfaction. This is a typical result of psychological scaling; a substantial increase of the current value is required to make the increase worthwhile. If $U(300) = 50$ and $U(1000) = 100$, then the parameters of the power function are $k = 1.9$ and $p = 0.6$: $U(S) = 1.9S^{0.6}$.

User preference for response time. There already exists some literature on user satisfaction for response time: the judgements of "acceptable" response times given by Shneiderman (1980, p 228: the times are taken from Miller, 1968). The times depend upon the task being performed. For highly interactive tasks, where the system has just changed state and the users are about to do a new action, 2.0 seconds seems appropriate. I determined that I would be twice as satisfied with a response time of 1 second. Therefore, $U(2\ sec) = 50$ and $U(1\ sec) = 100$. For these values, the power function becomes $U(T) = 100/T$ ($k = 100$, $= -1$).

Size of menu and display time. We need one more thing to complete the tradeoff analysis: the relationship between menu size (S) and the time to present the information (T). In general, time to present a display is a linear function of S: $T = \sigma + S/\beta$, where S is measured in characters, β is the display rate in characters per second (cps) and σ is system response time.

The tradeoff of menu size for display time. Knowledge of $U(S)$, $U(T)$, and the relationship between S and T, lets us determine the tradeoff between User Satisfaction for size of the menu and for time to display the information: $U(S)$ versus $U(T)$. If $\sigma << S/\beta$, then we can probably ignore σ, letting $T = S/\beta$. This lets us solve the tradeoff exactly. [1] If the two power functions are given by $U(S) = aS^p$ and $U(T) = bT^q$, then $U(S) = kU(T)^{p/q}$, where $k = \frac{a}{b^{p/q}}\beta^p$. The tradeoff relationship using the parameters estimated for menus is shown in Figure 1.

Maximizing Total User Satisfaction

Let overall satisfaction for the system, $U(system)$, be given by the weighted sum of the $U(x_i)$ values for each of its attributes, x_i: $U(system) = \sum \omega_i U(x_i)$, where ω_i is the weight for the i-*th* attribute. When there are only two attributes, x_1 and x_2, if we hold $U(system)$ constant at some value C, we can determine the *iso-satisfaction* line: $U(x_1) = \frac{C}{\omega_1} - \frac{\omega_2}{\omega_1}U(x_2)$. Thus, the iso-satisfaction functions appear on the tradeoff graphs as straight lines with a slope of $-\omega_2/\omega_1$, with higher lines representing higher values of $U(system)$.

If the tradeoff functions are concave downward (as are some in later figures), the maximum satisfaction occurs where the slope of the tradeoff function is tangent to the iso-satisfaction function: that is, when the slope of the tradeoff function = $-\omega_2/\omega_1$. In this case, maximum satisfaction occurs at some compromise between the two variables.

---

1. Letting $\sigma = 0$ simplifies the tradeoff relations, but this is not a necessary assumption. If system response time is slow, then σ should be reinstated: the tradeoff can still be be determined quite simply.

If the tradeoff functions are concave upward (as in Figure 1), then the *minimum* satisfaction occurs where the iso-satisfaction curves are tangent to the tradeoff functions. Maximum satisfaction occurs by maximizing one of the two attributes. The expert, for whom $\omega_T/\omega_S >> 1$, will not sacrifice time for a menu. The beginner, for whom $\omega_T/\omega_S << 1$, will sacrifice display time in order to get as big a menu as possible. For intermediate cases between that of the extreme expert or beginner, the optimum solution is still either to maximize menu size or to minimize display time, but the user might be indifferent as to which of these two was preferred. These conclusions apply to the tradeoff functions of Figure 1D regardless of display rate, as long as the curves are concave upward.[2]

These analyses say that the tradeoff solution that one tends to think of first — to compromise between time and menu size by presenting a small menu at some medium amount of workspace — actually provides the least amount of total satisfaction. Satisfaction is maximized by an all or *none solution*. The *all-or-none preference* applies only to tradeoff functions that are concave upward, such as that between menu size and display time. Later we shall see that when time is not relevant, the analysis of the tradeoff between menu size and workspace predicts that even experts will sacrifice some workspace for a menu.

Workspace

Available workspace refers to the amount of room left on the screen after the menu (or other information) is displayed. This is especially important where the menu stays on the screen while normal work continues. The tradeoff is sensitive to screen size. If we had a screen which could display 60 lines of text, using 6 lines to show the current state of the system and a small menu of choices would not decrease usability much. But if the screen could only display 3 lines at a time, then using 6 of them for this purpose would be quite detrimental.

User preference function for workspace. The user preference function for workspace clearly depends upon the nature of the task: some tasks — such as issuing a command — may only require a workspace of 1 line, others — such as file or text editing — could use unlimited workspace. Let us consider the workspace preferences for text editing of manuscripts. The most common editors can only show 24 lines, each of 80 characters: 1920 character positions. I let $U(1920) = 50$. To estimate the workspace that would double the value, I imagined working with screen editors of the sizes shown in Table 1. I concluded that I would need the size given by the two page journal spread. That is, $U(6400) = 100$. The power function parameters are $k = 0.64$ and $p = 0.6$, so that $U(w) = 0.64w^{0.6}$: the same exponent used for menu size but with a different scale factor, k. This function is shown in Figure 2.

Table 1

SIZE OF COMMON TEXTS AND DEVICES (in characters)	
TEXT OR DEVICE	APPROXIMATE NUMBER OF CHARACTER POSITIONS
Portable Computer (*Radio Shack Model 100*)	320
Home Microprocessor (*Apple II*)	960
Standard Video Display Unit	1,920
One typed manuscript page (*double spaced*)	2,600
One typed manuscript page (*single spaced*)	4,000
Journal page (*Cognitive Science*)	3,200
Double page spread	6,400
Page of Proceedings (*Gaithersberg Human Factors in Computer Systems*)	5,500
Double page spread	11,000
Newspaper page (*Los Angeles Times*)	30,000
Double page spread	60,000

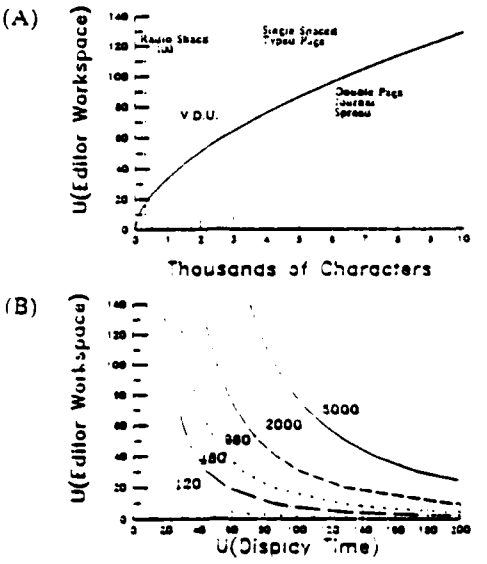


Figure 2. Panel A. User satisfaction function for editor workspace, w: $U(w) = 0.64w^{0.6}$. Typical character sizes for various displays and texts are also shown. Panel B shows the tradeoff function of workspace against time, $U(w)$ versus $U(T)$, for different display rates, β (*characters/second*). $U(T)$ is shown in Figure 1D: $U(T) = 100T^{-0.6}$

2. Whether a tradeoff function exhibits upward or downward concavity depends upon the choice of user satisfaction function. If both functions are power functions with one exponent positive and the other negative, the tradeoff functions are always concave upward. When both exponents are positive, the tradeoff functions are always concave downward. When the two functions are logarithmic, the tradeoff functions are always concave downward, and when they are both logistic, the tradeoff functions are both concave upward and downward, switching from one to the other as a function of the other variables (e.g., display rate). These conclusions hold whenever the two variables, x_1 and x_2, are linearly related.

Trading workspace for display time. One penalty for increasing the size of the workspace is increased time to display the workspace. If we use the same User Satisfaction function for time as in Figure 1B, we get the tradeoff functions shown in Figure 2B. Large workspaces require very high display rates before they are satisfactory. Because these tradeoff functions are concave upward (and are very similar to the functions of Figure 1D), the same conclusions apply here as to those earlier functions: the optimum operating point is an all-or-none solution. Thus, the user either prefers a large workspace, regardless of the time penalty, or a very fast display, regardless of the workspace penalty. Here, however, the relative weights are apt to be determined by the task rather than by the user's level of skill.

Suppose the task were one in which the display changes relatively infrequently. In this case we would expect $\omega_{workspace} >> \omega_{display\ rate}$, so that the optimum solution is to have as big a workspace as possible. If the task were one that requires frequent changes in the display, then we would expect the reverse result: $\omega_{workspace} << \omega_{display\ rate}$, so the optimum solution is to shrink the workspace to the smallest size at which the task can still be carried out, thereby minimizing display time.

Trading workspace for menu size. Adding a menu to the display decreases the amount of available workspace. Let W be the total size of the workspace that is available for use, w the workspace allocated to the text editor, and m the space allocated for a menu: $w = W - m$. We know that $U(m) = am^{\rho}$ and $U(w) = b(W-m)^{q}$, where $a = 2$, $b = 0.6$, and $\rho = q = 0.6$. This leads to the tradeoff functions shown in Figure 3.

Note that $U(editor\ workspace)$ is relatively insensitive to $U(menu\ size)$. This is because a relatively small sized display makes a satisfactory menu, whereas it requires a large display to make a satisfactory editor workspace. As a result, changing the size of the menu by only a few lines can make a large change in User Satisfaction, whereas the same change in workspace is usually of little consequence.

In some commercially available editors, the menu of commands can occupy approximately half the screen (usually 24 lines). Figure 1 indicates that for a menu of 12 lines (960 characters), $U(menu) = .00$. However, from Figures 2 and 3 we see that with a workspace of only 1920 characters, a menu of around 1000 characters (or of $U(menu) = .00$) reduces $U(editor\ workspace)$ from its value of .50 with no menu to .34: a reduction of almost one third. From Figure 3 we see that we would be much less impaired by the same size menu were the workspace considerably greater. In such cases, we have a clear tradeoff between the need for the menu information and the desire to have a reasonable workspace.

Maximizing total user satisfaction for menu and workspace. When tradeoff functions are concave downward (as in Figure 3), maximum satisfaction occurs where the slope of the tradeoff function is tangent to the iso-satisfaction function. For the user who values workspace and menu equally (so that the preferred slope is -1), the optimum solution is to operate at the right hand side of Figure 3. This makes for a relatively high value of user satisfaction for the menu (which means a large menu — the exact sizes can be determined from Figure 1A) — but with little sacrifice in user satisfaction for workspace. The more expert user will have an iso-satisfaction function with a much smaller slope, and so will sacrifice menu for workspace. Similarly, the beginner will have an iso-satisfaction curve with high slope which will maximize menu size at the expense of workspace. These results are quite unlike the tradeoffs that involved time in which an all-or-none solution was optimal: here, the optimum values are compromises between workspace and editor size. Display rate and amount of total available workspace alter the point of optimum operation. The analysis provides exact numerical determination of how the optimal operating point is affected by these variables.

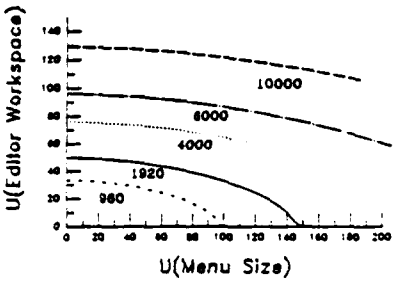


Figure 3. The tradeoff of User Satisfaction for menu size against User Satisfaction for editor workspace, for different values of total workspace, W. Horizontal lines represent constant values of $U(m)$ and, therefore, of m: the values of m can be determined from panel A of Figure 1. Similarly, vertical lines represent constant values of w: the values of w can be determined from panel A of Figure 2.

A Critique of the Tradeoff Analysis

There are a number of problems with the tradeoff analyses presented in this paper. There are two major criticisms, one minor one. Let me start with the minor one, for it represents a misunderstanding that would be good to clear up. I illustrate the misunderstanding for the variable of menu size, but the discussion applies to other variables as well:

- The functions must be wrong: *U(menu size)* continually increases as a function of *menu size*, yet when the size gets too large, the menu becomes less useful: *U(menu size)* should also decrease.

User Satisfaction for the System Is the Sum of Its Parts

This misunderstanding derives from confusing *User Satisfaction* for a single attribute with *User Satisfaction* for a system. A major philosophy of the tradeoffs analysis is that a system can be decomposed into its underlying component attributes and *User Satisfaction* for each assessed individually. The *User Satisfaction* for the entire system can only be determined from the combination of the *User Satisfaction*

values for each of its components. The satisfaction for the amount of information conveyed by the menu continues to increase with size, but that the ability to find something (captured by "search time") decreases with increasing size of a menu. The overall satisfaction for the menu is given by the sum of the increasing satisfaction for the information and the decreasing satisfaction for search effort: the result is a U-shaped curve that decreases as size gets too big.

Now let me address the two major critiques:

- The tradeoff functions are arbitrary;
- How do we determine the functions when we design? It would be more useful were there a set of standards (perhaps in handbook form);

These two issues point to unsolved problems with the method. My defense is to argue that this procedure is new. The goal is to introduce the philosophy and to encourage others to help in the collection of the relevant data and in the development of the method. However, the numbers and the particular functions used here may be useful, for the tasks for which they were derived: they do mesh well with my intuitions.

The Tradeoff Functions Are Arbitrary

Although the functions used here are indeed arbitrary, three things need to be noted. First, power functions have a long tradition of satisfactory use in psychology and so are apt to be good approximations. Second, I have actually computed User Satisfaction functions using the logistic, power, and logarithmic functions; over much of the range of interest, the results differed suprisingly little, although at high data rates, the concave upward tradeoff functions became concave downward when the logistic was used for size and time, although at low data rates they were still concave upward (see footnote 2). Third, I agree that the preferred thing would be to have an experimental program to determine the exact forms and parameters of the functions. In particular, the all-or-none prediction is sensitive to the form of the User Satisfaction functions.

How Do We Determine the Functions When We Design?

Here, again, empirical work is needed. I suspect the functions will be found to vary only for a reasonably small number of classes of users, classes of tasks, and design attributes, so that it would be possible to collect typical values in a handbook. Alternatively, quick data collection methods might be devised: the magnitude estimation procedures are especially easy to apply. A handbook might be quite valuable. Before this can be done, of course, it is necessary to determine that the hypothesis is correct — that there are a relatively small number of tasks, user classes, and tradeoff variables that need be considered. Moreover, one must extend the analysis to a larger domain of problems and demonstrate its usefulness in actual design.

Some Other Examples of Tradeoffs

There are numerous other tradeoff analyses in addition to the ones presented here. Three other situations seem important enough to warrant consideration here, even though they are not yet ready for quantitative treatment. These are: (1) the comparison between command languages and menu-driven systems; (2) how to choose names for commands and files; and (3), the tradeoffs that result when moving among computer systems of widely varying capabilities, as in the differences between hand-held computers and powerful, networked workstations.

Command Languages Versus Menus

The relative merits of menu-based systems and command language systems are often debated, seldom with any firm conclusion. It is useful to compare their tradeoffs, but before we do, it is necessary to be clear about what is meant by each alternative. In this context a command language system is one in which no aids are presented to the user during the intention or choice stages, and the action specification is performed by typing a command, using the syntax required by the operating system. (The distinctions among the intention, choice, and specification stages come from Norman, 1983b.) Command languages are the most frequent method of implementing operating systems. Similarly, in this context a menu-based system is one in which all commands are presented via menus, where a command cannot be specified unless it is currently being shown on the active menu, and where the commands are specified either through short names or single characters (as indicated by the menu items) or by pointing at the relevant menu item (or perhaps at a switch or mark indicated by the item). These are restricted interpretations of the two alternatives, confounding issues about the format for information presentation and action specification. Still, because they represent common design alternatives, it is useful to compare them.

Command languages offer experts great versatility. Because of their large amount of knowledge and experience with the system, experts tend to know exactly what operations they wish performed. With a command language they can specify their operations directly simply by typing the names of the commands, as well as any parameters, files, or other system options that are required. Command languages make it easy to specify parameters (or "flags") to commands, something that may be difficult with menu-based systems.

Menus offer the beginner or the casual user considerable assistance. At any stage, information is available. Even abbreviated menus serve as a reminder of the alternatives. Experts often complain about menu-based systems because of the time penalty in requesting a menu, waiting for it to be displayed, and then searching for the desired item. Moreover, systems with large numbers of commands require multiple menus that slow up the expert. The problem is that the system is designed to give help, whether or not the user wishes it.

Two of the difficulties with menus are the delay in waiting for them to be plotted and the amount of space they occupy. Figure 1D shows that the tradeoff between amount of information and time delay is especially sensitive to information transmission rate. When transmission time becomes fast enough, there is little penalty for menus, whereas at slow rates of data transmission, the penalty is high. In similar fashion, Figure 3 shows that the tradeoff between menu size and workspace is especially sensitive to the amount of total workspace available. When sufficient workspace is available, there is little penalty for menus. Thus, slow transmission rates and small workspaces bias the design choice toward command language systems; high data rates and large workspaces bias the system toward menu-based systems.

The two systems also differ in the kinds of errors they lead to and ease of error correction. In a command language system, an error in command specification usually leads to an illegal command: no action gets performed. This error is usually easy to detect and to correct. In a menu-based system, an error in specification is almost always a legal command. This error can be very difficult to correct. If the action was subtle, the user may not even be aware it was performed. If the action was dramatic, the user will often have no idea of what precipitated it, since the action specification was unintentional.

Some of the tradeoffs associated with menu-based systems and command language systems are summarized in Table 2. Command languages tend to be virtuous for the expert, but difficult for the novice; they are difficult to learn and there are no on-line reminders of the set of possible actions. Menus are easy to use and they provide a constant reminder. On the other hand, menus tend to be slow — for some purposes, the expert finds them tedious and unwieldy — and not as flexible as command languages.

This analysis is brief and restricted to the particular formats of command language and menu-based systems that were described. There do exist techniques for mitigating the deficiencies of each system. Nonetheless, the analysis is useful, both for pointing out the nature of the issues and for being reasonably faithful to some existing systems. In the argument over which system is best, the answer must be that neither is: each has its virtues and its deficiencies.

The Choice of Names for Commands and Files

Another example of a common tradeoff is in the choice of name for a command or a file. The problem occurs because the name must serve two different purposes: as a *description* of the item and also as the string of characters that must be typed to invoke it, that is, as the *specification*; these two uses pose conflicting requirements.

Consider the properties of names when used as descriptions. The more complete the description, the more useful it can be, especially when the user is unsure of the options or is selecting from an unfamiliar set of alternatives. However, the longer the description, the more space it occupies and the more difficult to read or scan the material. In addition, there are often system limitations on the length and format of names. For these reasons, one usually settles for a partial description, counting on context or prior knowledge to allow the full description to be regenerated by the user.

Table 2

TRADEOFFS BETWEEN MENU-BASED SYSTEMS AND COMMAND LANGUAGE SYSTEMS

ATTRIBUTE	MENU-BASED	COMMAND LANGUAGE
Speed of use:	Slow, especially if large or if has hierarchical structure.	Fast, for experts; operation can be specified exactly, regardless of system state.
Prior knowledge required.	Very little — can be self-explanatory.	Considerable — user is expected to have learned set of alternative actions and command language that specifies them.
Ease of learning:	High. Uses recognition memory: easier and more accurate than recall memory. Easy to explore system and discover options.	Low. Users must learn names and syntax of language. If alternatives are numerous, learning may take considerable time. No simple way to explore system and discover options not already known.
Errors:	Specification error leads to inappropriate action: difficult to determine what happened and to correct.	Specification error usually leads to illegal command: easy to detect, easy to correct.
Most useful for:	Beginner or infrequent user.	Expert or frequent user.

Once the appropriate name has been determined, the user enters the specification stage of operation; the user must specify to the computer system which name is desired. Most users are not expert typists, and so it is desirable to simplify the specification stage. As a result, there is pressure toward the use of short names, oftentimes to the limit of single character command names. [3]

The desirability for short names is primarily a factor when specification must be done by naming. When the specification can be done by pointing, then ease of typing is no longer a factor. Nonetheless, there are still constraints on the name choice: the longer the name, the easier to find and point at the desired item, but at the cost of using a larger percentage of the available workspace, of increasing display time, and the ease of reading and search. Now names might wish to be chosen so they are visually distinct, or so that they occupy appropriate spatial locations on the display, in all cases adding more constraints to the naming

3. A number of systems allow for shortcuts in specification, so that one need only use sufficient characters (plus some "escape" or "wild-card" character) to make the name unique. This option poses its own naming constraints; now a name is chosen not only to be descriptive, but with the added requirement that one or two letters be sufficient to distinguish it uniquely from all other names. The typing aid introduces its own form of naming constraint.

problem. In general the descriptive requirements tend to push toward longer names, names that provide as much information as possible. The specification requirements tend to push toward shorter names, names that are easy to type.

Handheld Computers Versus Workstations

New developments in technology are moving computer systems in several conflicting directions simultaneously. Workstations are getting more powerful, with large memories, large, high resolution screens, and with very high communication bandwidths. These developments move us toward the ability to present as much information as is needed by the user with little penalty in time, workspace, or even memory space. At the same time, some machines are getting smaller, providing us with briefcase sized and handheld computers. These machines have great virtue because of their portability, but severe limitations in communications speed, memory capacity, and amount of display screen or workspace.

Just as workstations are starting to move toward displays capable of 1000 line resolution, showing several entire pages of text, handheld computers move us back toward only a few short lines — perhaps 8 lines of 40 characters each — and communication rates of 30 cps (300 baud). The major differences between workstations and handheld computers relevant to the tradeoffs discussion are in the amount of memory, processor speed and power, communication abilities, availability of extra peripherals, and screen size: in all cases, the handheld machine has sacrificed power for portability. Because the same people may wish to use both handheld machines and workstations (one while at home or travelling, the other at work), the person may wish the same programs to operate on the two machines. However, the interface design must be different, as the tradeoff analyses of this paper show.

Summary and Conclusions

The tradeoff analysis is intended to serve as an example of a quantitative design tool. In some cases it may not be possible to select an optimum design, not even for a restricted class of activities and users. In these cases, knowledge of the tradeoffs allows the designer to choose intelligently, knowing exactly what benefits and limits the system design will provide. Finally, the analyses show that some design decisions are heavily affected by technology, others are not. Thus, answers to design questions are heavily context dependent, being affected by the classes of users for whom the system is intended, the types of applications being performed, which stages of user activities are thought to be of most importance, and the level of technology being employed.

The work presented here is just the beginning. In the ideal case, the tradeoff relationships will be known exactly, perhaps with the relevant quantitative parameters provided in handbooks. This paper has limited itself to demonstrating the basic principles. Considerable development must still be done on this issue and on the other major parameters and issues that affect the quality of the human-machine interaction. Much work remains to be done.

A second point of the paper is to argue for more fundamental approaches to the study of human-machine interaction. All too often we are presented with minor studies that do not lead to general application, or to studies that are restricted to a particular technology. All too often we are trapped in the tar pits of the field or seduced by the sirens of technology. If we are to have a science of design that can be of use beyond today's local problems, we must learn to broaden our views, sharpen our methods, and avoid temptation.

A major moral of this paper is that it is essential to analyze separately the different aspects of human-computer interaction. Detailed analyses of each aspect of the human-computer interface are essential, of course, but because design decisions interact across stages and classes of users, we must also develop tools that allow us to ask for what purpose the system is to be used, then to determine how best to accomplish that goal. Only after the global decisions have been made should the details of the interface design be determined.

References

Bannon, L., Cypher, A., Greenspan, S., & Monty, M.L. Evaluation and analysis of users' activity organization. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston, December, 1983.

Card, S., Moran, T., & Newell, A., *Applied Information-Processing Psychology: The Human-Computer* Interface. Hillsdale, N.J.: Erlbaum Associates, 1983.

Miller, R. B. Response time in man-computer conversational transactions, *Proceedings of the Spring Joint Computer Conference*, 1968, *33*. Montvale, New Jersey, pp. 267-277.

Norman, D. A. Design rules based on analyses of human error. *Communications of the ACM*, 1983a, *4*, 254-258.

Norman, D. A. Four stages of user activities. Manuscript. 1983b.

Norman, D. A. Tradeoffs in the design of human-computer interfaces. Manuscript. 1983c.

O'Malley, C., Smolensky, P., Bannon, L., Conway, E., Graham, J., Sokolov, J., & Monty, M. L. A proposal for user centered system documentation. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston, December, 1983.

Root, R. W., & Draper, S. Questionnaires as a software evaluation tool. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems*. Boston, December, 1983.

Shneiderman, B. *Software Psychology: Human Factors in Computer and Information* Systems. Cambridge, Mass.: Winthrop Publishers, 1980.

Stevens, S. S. Perceptual magnitude and its measurement. In E. C. Carterette & M. P. Friedman (Eds.), *Handbook of perception* (Vol. 2). New York: Academic Press, 1974.

Cognitive Science ONR Technical Report List

8001. Donald R. Gentner, Jonathan Grudin, and Eileen Conway. *Finger Movements in Transcription Typing*. May 1980.

8002. James L. McClelland and David E. Rumelhart. *An Interactive Activation Model of the Effect of Context in Perception: Part I*. May 1980.

8003. David E. Rumelhart and James L. McClelland. *An Interactive Activation Model of the Effect of Context in Perception: Part II*. July 1980.

8004. Donald A. Norman. *Errors in Human Performance*. August 1980.

8005. David E. Rumelhart and Donald A. Norman. *Analogical Processes in Learning*. September 1980.

8006. Donald A. Norman and Tim Shallice. *Attention to Action: Willed and Automatic Control of Behavior*. December 1980.

8101. David E. Rumelhart. *Understanding Understanding*. January 1981.

8102. David E. Rumelhart and Donald A. Norman. *Simulating a Skilled Typist: A Study of Skilled Cognitive-Motor Performance*. May 1981.

8103. Donald R. Gentner. *Skilled Finger Movements in Typing*. July 1981.

8104. Michael I. Jordan. *The Timing of Endpoints in Movement*. November 1981.

8105. Gary Perlman. *Two Papers in Cognitive Engineering: The Design of an Interface to a Programming System and MENUNIX: A Menu-Based Interface to UNIX (User Manual)*. November 1981.

8106. Donald A. Norman and Diane Fisher. *Why Alphabetic Keyboards Are Not Easy to Use: Keyboard Layout Doesn't Much Matter*. November 1981.

8107. Donald R. Gentner. *Evidence Against a Central Control Model of Timing in Typing*. December 1981.

8201. Jonathan T. Grudin and Serge Larochelle. *Digraph Frequency Effects in Skilled Typing*. February 1982.

8202. Jonathan T. Grudin. *Central Control of Timing in Skilled Typing.* February 1982.

8203. Amy Geoffroy and Donald A. Norman. *Ease of Tapping the Fingers in a Sequence Depends on the Mental Encoding.* March 1982.

8204. LNR Research Group. *Studies of Typing from the LNR Research Group: The role of context, differences in skill level, errors, hand movements, and a computer simulation.* May 1982.

8205. Donald A. Norman. *Five Papers on Human-Machine Interaction.* May 1982.

8206. Naomi Miyake. *Constructive Interaction.* June 1982.

8207. Donald R. Gentner. *The Development of Typewriting Skill.* September 1982.

8208. Gary Perlman. *Natural Artificial Languages: Low-Level Processes.* December 1982.

8301. Michael C. Mozer. *Letter Migration in Word Perception.* April 1983.

8302. David E. Rumelhart and Donald A. Norman. *Representation in Memory.* June 1983.

8303. The HMI Project at University of California, San Diego. *User Centered System Design: Papers for the CHI '83 Conference on Human Factors in Computer Systems.* November 1983.

ICS Technical Report List

8301. David Zipser. *The Representation of Location*. May 1983.

8302. Jeffrey Elman & Jay McClelland. *Speech Perception as a Cognitive Process: The Interactive Activation Model*. April 1983.

8303. Ron Williams. *Unit Activation Rules for Cognitive Networks*. November 1983.

8304. David Zipser. *The Representation of Maps*. November 1983.

8305. The HMI Project. *User Centered System Design: Papers for the CHI '83 Conference on Human Factors in Computer Systems*. November 1983.

Navy

1 Robert Ahlers
Code N711
Human Factors Laboratory
NAVTRAEQUIPCEN
Orlando, FL 32813

1 Dr. Ed Aiken
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Arthur Bachrach
Environmental Stress Program Center
Naval Medical Research Institute
Bethesda, MD 20014

1 Dr. Alvah Bittner
Naval Biodynamics Laboratory
New Orleans, LA 70189

1 Code N711
Attn: Arthur S. Blaiwes
Naval Training Equipment Center
Orlando, FL 32813

1 Dr. Robert Blanchard
Navy Personnel R&D Center
San Diego, CA 92152

1 Liaison Scientist
Office of Naval Research
Branch Office, London
Box 39
FPO New York, NY 09510

1 Chief of Naval Education and Training
Liason Office
Air Force Human Resource Laboratory
Operations Training Division
WILLIAMS AFB, AZ 85224

1 Dr. Stanley Collyer
Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217

1 CDR Mike Curran
Office of Naval Research
900 N. Quincy St.
Code 270
Arlington, VA 22217

Navy

1 DR. PAT FEDERICO
Code P13
NPRDC
San Diego, CA 92152

1 Dr. Jude Franklin
Code 7510
Navy Research Laboratory
Washington, DC 20375

1 Dr. Mike Gaynor
Navy Research Laboratory
Code 7510
Washington, DC 20375

1 LT Steven D. Harris, MSC, USN
RFD 1, Box 243
Riner, VA 24149

1 Dr. Jim Hollan
Code 14
Navy Personnel R & D Center
San Diego, CA 92152

1 Dr. Ed Hutchins
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. Peter Kincaid
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. James Lester
ONR Detachment
495 Summer Street
Boston, MA 02210

1 Dr. William L. Maloy (02)
Chief of Naval Education and Training
Naval Air Station
Pensacola, FL 32508

1 CAPT Richard L. Martin, USN
Commanding Officer
USS Carl Vinson (CVN-70)
FPO New York, NY 09558

Navy

1 Dr. Joe McLachlan
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. George Moeller
Director, Behavioral Sciences Dept.
Naval Submarine Medical Research Lab
Naval Submarine Base
Groton, CT 06349

1 Dr William Montague
NPRDC Code 13
San Diego, CA 92152

1 Technical Director
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 Office of Naval Research
Code 433
800 N. Quincy SStreet
Arlington, VA 22217

6 Personnel & Training Research Group
Code 442PT
Office of Naval Research
Arlington, VA 22217

1 Office of the Chief of Naval Operations
Research Development & Studies Branch
OP 115
Washington, DC 20350

1 Dr. Gary Poock
Operations Research Department
Code 55PK
Naval Postgraduate School
Monterey, CA 93940

1 Dr. Gil Ricard
Code N711
NTEC
Orlando, FL 32813

1 Dr. Robert G. Smith
Office of Chief of Naval Operations
OP-987H
Washington, DC 20350

Navy

1 Dr. Alfred F. Smode, Director
Training Analysis & Evaluation Group
Dept. of the Navy
Orlando, FL 32813

1 Dr. Richard Sorensen
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Frederick Steinheiser
CNO - OP115
Navy Annex
Arlington, VA 20370

1 Roger Weissinger-Baylon
Department of Administrative Sciences
Naval Postgraduate School
Monterey, CA 93940

1 Mr John H. Wolfe
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Wallace Wulfeck, III
Navy Personnel R&D Center
San Diego, CA 92152

Marine Corps

1 M. William Greenup
Education Advisor (E031)
Education Center, MCDEC
Quantico, VA 22134

1 Special Assistant for Marine Corps Matters
Code 100M
Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217

1 DR. A.L. SLAFKOSKY
SCIENTIFIC ADVISOR (CODE RD-1)
HQ, U.S. MARINE CORPS
WASHINGTON, DC 20380

Army

1 Technical Director
U. S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Beatrice J. Farr
U. S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Milton S. Katz
Training Technical Area
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Marshall Narva
US Army Research Institute for the Behavioral & Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Harold F. O'Neil, Jr.
Director, Training Research Lab
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Commander, U.S. Army Research Institute for the Behavioral & Social Sciences
ATTN: PERI-BR (Dr. Judith Orasanu)
5001 Eisenhower Avenue
Alexandria, VA 20333

1 Joseph Psotka, Ph.D.
ATTN: PERI-IC
Army Research Institute
5001 Eisenhower Ave.
Alexandria, VA 22333

1 Dr. Robert Sasmor
U. S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. Robert Wisher
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Air Force

1 Technical Documents Center
Air Force Human Resources Laboratory
WPAFB, OH 45433

1 U.S. Air Force Office of Scientific Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, DC 20332

1 Air University Library
AUL/LSE 76/443
Maxwell AFB, AL 36112

1 Dr. Earl A. Alluisi
HQ, AFHRL (AFSC)
Brooks AFB, TX 78235

1 Mr. Raymond E. Christal
AFHRL/MOE
Brooks AFB, TX 78235

1 Bryan Dallman
AFHRL/LRT
Lowry AFB, CO 80230

1 Dr. Alfred R. Fregly
AFOSR/NL
Bolling AFB, DC 20332

1 Dr. Genevieve Haddad
Program Manager
Life Sciences Directorate
AFOSR
Bolling AFB, DC 20332

1 Dr. T. M. Longridge
AFHRL/OTE
Williams AFB, AZ 85224

1 Dr. John Tangney
AFOSR/NL
Bolling AFB, DC 20332

1 Dr. Joseph Yasutake
AFHRL/LRT
Lowry AFB, CO 80230

Department of Defense

12 Defense Technical Information Center
Cameron Station, Bldg 5
Alexandria, VA 22314
Attn: TC

1 Military Assistant for Training and Personnel Technology
Office of the Under Secretary of Defens for Research & Engineering
Room 3D129, The Pentagon
Washington, DC 20301

1 Major Jack Thorpe
DARPA
1400 Wilson Blvd.
Arlington, VA 22209

Civilian Agencies

1 Dr. Patricia A. Butler
NIE-BRN Bldg. Stop # 7
1200 19th St., NW
Washington, DC 20208

1 Dr. Susan Chipman
Learning and Development
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Dan Lyon
AFHRL/OT (NRAI)
Williams AFB, AZ 85225

1 Dr. Arthur Melmed
724 Brown
U. S. Dept. of Education
Washington, DC 20208

1 Dr. Andrew R. Molnar
Office of Scientific and Engineering
Personnel and Education
National Science Foundation
Washington, DC 20550

1 Everett Palmer
Research Scientist
Mail Stop 239-3
NASA Ames Research Center
Moffett Field, CA 94035

1 Dr. Mary Stoddard
C 10, Mail Stop B296
Los Alamos National Laboratories
Los Alamos, NM 87545

1 Chief, Psychological Research Branch
U. S. Coast Guard (G-P-1/2/TP42)
Washington, DC 20593

1 Dr. Edward C. Weiss
National Science Foundation
1800 G Street, NW
Washington, DC 20550

1 Dr. Frank Withrow
U. S. Office of Education
400 Maryland Ave. SW
Washington, DC 20202

Civilian Agencies

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

Private Sector

1 Dr. John R. Anderson
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. John Annett
Department of Psychology
University of Warwick
Coventry CV4 7AJ
ENGLAND

1 Dr. Michael Atwood
ITT - Programming
1000 Oronoque Lane
Stratford, CT 06497

1 1 Psychological Research Unit
NBAH-3-44 Attn: Librarian
Northbourne House
Turner ACT 2601
AUSTRALIA

1 Dr. Alan Baddeley
Medical Research Council
Applied Psychology Unit
15 Chaucer Road
Cambridge CB2 2EF
ENGLAND

1 Dr. Patricia Baggett
Department of Psychology
University of Colorado
Boulder, CO 80309

1 Dr. Jonathan Baron
80 Glenn Avenue
Berwyn, PA 19312

1 Mr. Avron Barr
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Menucha Birenbaum
School of Education
Tel Aviv University
Tel Aviv, Ramat Aviv 69978
Israel

1 Dr. John Black
Yale University
Box 11A, Yale Station
New Haven, CT 06520

Private Sector

1 Dr. John S. Brown
XEROX Palo Alto Research Center
3333 Coyote Road
Palo Alto, CA 94304

1 Dr. Glenn Bryan
6208 Poe Road
Bethesda, MD 20817

1 Dr. Bruce Buchanan
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Bundesministerium der Verteidigung
-Referat P II 4-
Psychological Service
Postfach 1328
D-5300 Bonn 1
F. R. of Germany

1 Dr. Jaime Carbonell
Carnegie-Mellon University
Department of Psychology
Pittsburgh, PA 15213

1 Dr. Pat Carpenter
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. William Chase
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Micheline Chi
Learning R & D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15213

1 Dr. William Clancey
Department of Computer Science
Stanford University
Stanford, CA 94306

1 Dr. Michael Cole
University of California
at San Diego
Laboratory of Comparative
Human Cognition - D003A
La Jolla, CA 92093

Private Sector

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 Dr. Kenneth B. Cross
Anacapa Sciences, Inc.
P.O. Drawer Q
Santa Barbara, CA 93102

1 ERIC Facility-Acquisitions
4833 Rugby Avenue
Bethesda, MD 20014

1 Professor Reuven Feuerstein
HWCRI Rehov Karmon 6
Bet Hakerem
Jerusalem
Israel

1 Mr. Wallace Feurzeig
Department of Educational Technology
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02238

1 Univ. Prof. Dr. Gerhard Fischer
Liebiggasse 5/3
A 1010 Vienna
AUSTRIA

1 Dr. Dexter Fletcher
WICAT Research Institute
1875 S. State St.
Orem, UT 22333

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 Dr. Michael Genesereth
Department of Computer Science
Stanford University
Stanford, CA 94305

1 Dr. Don Gentner
Center for Human Information Processing
University of California, San Diego
La Jolla, CA 92093

Private Sector

1 Dr. Dedre Gentner
Bolt Beranek & Newman
10 Moulton St.
Cambridge, MA 02138

1 Dr. Robert Glaser
Learning Research & Development Center
University of Pittsburgh
3939 O'Hara Street
PITTSBURGH, PA 15260

1 Dr. Joseph Goguen
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

1 Dr. Daniel Gopher
Department of Psychology
University of Illinois
Champaign, IL 61820

1 DR. JAMES G. GREENO
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Barbara Hayes-Roth
Department of Computer Science
Stanford University
Stanford, CA 95305

1 Dr. Frederick Hayes-Roth
Teknowledge
525 University Ave.
Palo Alto, CA 94301

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98105

1 Dr. Marcel Just
Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Steven W. Keele
Dept. of Psychology
University of Oregon
Eugene, OR 97403

Private Sector

1 Dr. David Kieras
Department of Psychology
University of Arizona
Tuscon, AZ 85721

1 Dr. Walter Kintsch
Department of Psychology
University of Colorado
Boulder, CO 80302

1 Dr. Stephen Kosslyn
1236 William James Hall
33 Kirkland St.
Cambridge, MA 02138

1 Dr. Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

1 Dr. Jill Larkin
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213

1 Dr. Alan Lesgold
Learning R&D Center
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 15260

1 Dr. Jim Levin
University of California
at San Diego
Laboratory fof Comparative
Human Cognition - D003A
La Jolla, CA 92093

1 Dr. Michael Levine
Department of Educational Psychology
210 Education Bldg.
University of Illinois
Champaign, IL 61801

1 Dr. Jay McClelland
Department of Psychology
MIT
Cambridge, MA 02139

1 Dr. James R. Miller
Computer*Thought Corporation
1721 West Plano Highway
Plano, TX 75075

Private Sector

1 Dr. Mark Miller
Computer*Thought Corporation
1721 West Plano Parkway
Plano, TX 75075

1 Dr. Tom Moran
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 Dr. Allen Munro
Behavioral Technology Laboratories
1845 Elena Ave., Fourth Floor
Redondo Beach, CA 90277

1 Committee on Human Factors
JH 811
2101 Constitution Ave. NW
Washington, DC 20418

1 Dr. Jesse Orlansky
Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311

1 Dr. Nancy Pennington
University of Chicago
Graduate School of Business
1101 E. 58th St.
Chicago, IL 60637

1 DR. PETER POLSON
DEPT. OF PSYCHOLOGY
UNIVERSITY OF COLORADO
BOULDER, CO 80309

1 Dr. Fred Reif
Physics Department
University of California
Berkeley, CA 94720

1 Dr. Lauren Resnick
LRDC
University of Pittsburgh
3939 O'Hara Street
Pittsburgh, PA 1521

1 Mary S. Riley
Program in Cognitive Science
Center for Human Information Processing
University of California, San Diego
La Jolla, CA 92093

Private Sector

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Ernst Z. Rothkopf
Bell Laboratories
Murray Hill, NJ 07974

1 Dr. William B. Rouse
Georgia Institute of Technology
School of Industrial & Systems
Engineering
Atlanta, GA 30332

1 Dr. David Rumelhart
Center for Human Information Processing
Univ. of California, San Diego
La Jolla, CA 92093

1 Dr. Michael J. Samet
Perceptronics, Inc
6271 Variel Avenue
Woodland Hills, CA 91364

1 Dr. Roger Schank
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Walter Schneider
Psychology Department
603 E. Daniel
Champaign, IL 61820

1 Mr. Colin Sheppard
Applied Psychology Unit
Admiralty Marine Technology Est.
Teddington, Middlesex
United Kingdom

1 Dr. H. Wallace Sinaiko
Program Director
Manpower Research and Advisory Services
Smithsonian Institution
801 North Pitt Street
Alexandria, VA 22314

1 Dr. Edward E. Smith
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

Private Sector

1 Dr. Richard Snow
School of Education
Stanford University
Stanford, CA 94305

1 Dr. Elliott Soloway
Yale University
Department of Computer Science
P.O. Box 2158
New Haven, CT 06520

1 Dr. Kathryn T. Spoehr
Psychology Department
Brown University
Providence, RI 02912

1 Dr. Robert Sternberg
Dept. of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 Dr. Albert Stevens
Bolt Beranek & Newman, Inc.
10 Moulton St.
Cambridge, MA 02238

1 David E. Stone, Ph.D.
Hazeltine Corporation
7680 Old Springhouse Road
McLean, VA 22102

1 Dr. Kikumi Tatsuoka
Computer Based Education Research Lab
252 Engineering Research Laboratory
Urbana, IL 61801

1 Dr. Maurice Tatsuoka
220 Education Bldg
1310 S. Sixth St.
Champaign, IL 61820

1 Dr. Perry W. Thorndyke
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 Dr. Douglas Towne
Univ. of So. California
Behavioral Technology Labs
1845 S. Elena Ave.
Redondo Beach, CA 90277

Private Sector

1 Dr. Kurt Van Lehn
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 Dr. Keith T. Wescourt
Perceptronics, Inc.
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

1 William B. Whitten
Bell Laboratories
20-610
Holmdel, NJ 07733

1 Dr. Christopher Wickens
Department of Psychology
University of Illinois
Champaign, IL 61820

1 Dr. Mike Williams
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

1 Dr. Joseph Wohl
Alphatech, Inc.
3 New England Executive Park
Burlington, MA 01803